



#### MACHINE LEARNING LAB [PC651CS] LABORATORY MANUAL

***For***

#### B.E - VI Semester Prepared By Dr. U. Moulali

**Associate Professor**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

***Empower youth- Architects of Future World***



**VISION**

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

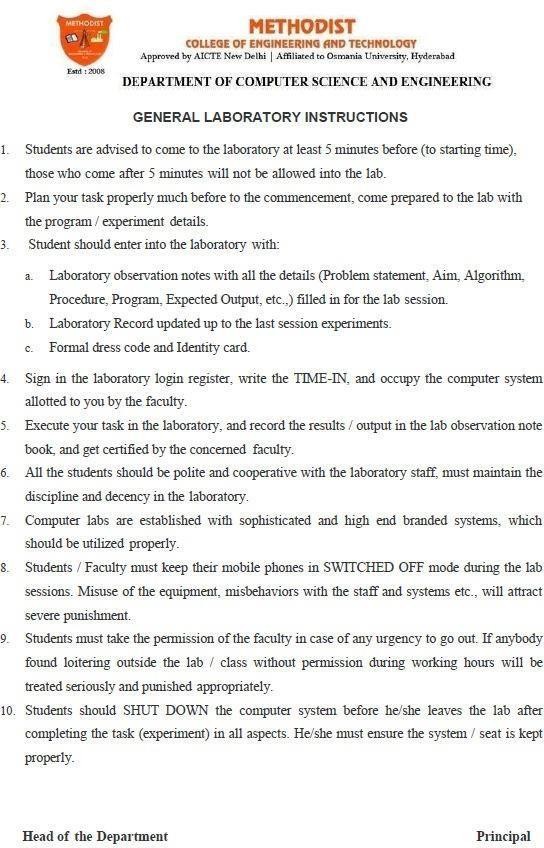
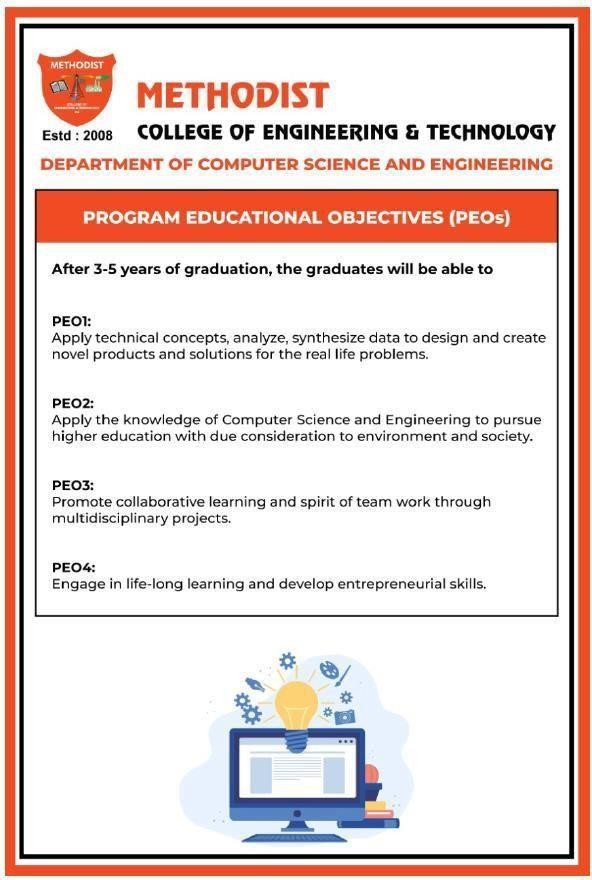
# MISSION

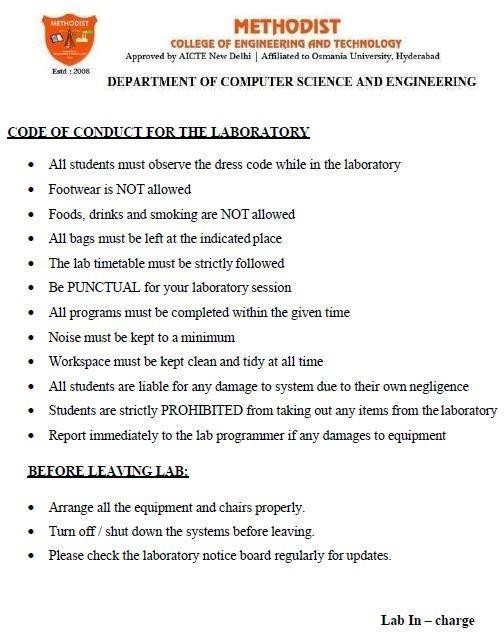
To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.









### LIST OF COS

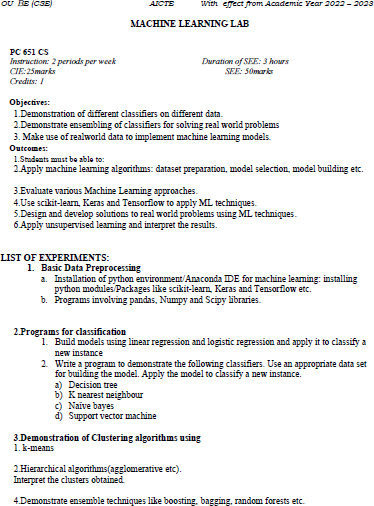
**MACHINE LEARNINGLAB (PC651CS)**

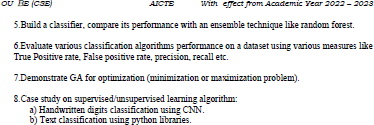
|  |  |  |
| --- | --- | --- |
| **Course Code** | **Course Outcome** | **Taxonomy Level** |
| PC651CS.1 | Familiarizing with Anaconda and Jupiter for importing modules and dependencies for ML | Applying |
| PC651CS.2 | Apply machine learning algorithms: dataset preparation, model selection, model building etc | Evaluating |
| PC651CS.3 | Evaluate various Machine Learning approaches | Applying |
| PC651CS.4 | Use scikit-learn, Keras and Tensorflow to apply ML techniques | Analyzing |
| PC651CS.5 | Design and develop solutions to real world problems using ML techniques | Creating |
| PC651CS.6 | Apply unsupervised learning and interpret the results. | Creating |

### ML Lab (PC651CS) CO-PO Mapping

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PO / CO** | **P O 1** | **P O 2** | **P O 3** | **P O 4** | **P O 5** | **P O 6** | **P O 7** | **P O 8** | **P O 9** | **P O 10** | **P O 11** | **P O 12** | **P S 01** | **PS O 2** | **PS O 3** |
| **PC651CS.1** | **3** |  | **2** | **2** | **3** |  |  |  | **1** | **2** |  | **1** | **3** | **2** | **1** |
| **PC651CS.2** | **2** | **3** | **3** | **3** | **3** |  |  |  | **2** | **2** | **1** | **2** | **3** | **2** | **2** |
| **PC651CS.3** | **2** | **3** | **3** | **3** | **2** |  |  |  | **2** | **2** | **1** | **1** | **3** | **2** | **3** |
| **PC651CS.4** | **1** | **1** | **3** | **2** | **3** |  |  |  | **2** | **2** | **1** | **1** | **2** | **2** | **3** |
| **PC651CS.5** | **3** |  | **2** | **1** | **2** |  |  |  | **2** | **2** | **1** | **2** | **2** | **2** | **2** |
| **PC651CS.6** | **3** | **1** | **1** | **3** | **3** |  |  |  | **2** | **2** | **1** | **1** | **2** | **2** | **2** |
| **PC651CS** | **2.33** | **2.00** | **2.33** | **2.33** | **2.67** |  |  |  | **1.83** | **2.00** | **1.00** | **1.33** | **2.50** | **2.0** | **2.2** |









|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp No.** | **Name of the Experiment** | **Date of Conducted** | **Date of Submission** | **Faculty Signature** |
| 1. | Introduction to Python programming and various libraries used for machine learning. |  |  |  |
| 2 | For a given set of data write a program to implement Linear Regression algorithm to create a model and evaluate the model. |  |  |  |
| 3 | Write a python program to classify the given dataset using Logistic Regression and evaluate the model. |  |  |  |
| 4 | Write a program to demonstrate the working of the Decision Tree based model classification. |  |  |  |
| 5. | Write a program to implement the Naïve Bayesian classifier for a sample training data set. Compute the accuracy of the classifier, considering a few test data sets. |  |  |  |
| 6. | Write a python program to implement the Principal component analysis (PCA) Algorithm in a given Dataset. |  |  |  |
| 7. | Write a python program to implement the k-Means Algorithm using the appropriate dataset. |  |  |  |
| 8 | Write a python program to implement the k-Nearest Neighbor Algorithm using the appropriate dataset. |  |  |  |
| 9 | Write a program to implement Support Vector Machine in python. |  |  |  |
| 10 | Demonstrate ensemble – bagging based classification |  |  |  |
| 11 | Demonstrate ensemble - random forest based classification |  |  |  |
| 12 | Evaluate any two classification algorithms  performance on a dataset using various measures like True Positive rate, false positive rate, precision, recall etc. |  |  |  |
|  | **ADDITIONAL EXPERIMENTS** |  |  |  |
| 1 | Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program |  |  |  |
| 2 | Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets |  |  |  |



LIST OF EXPERIMENTS:

#### Basic Data Preprocessing

* 1. Installation of python environment/Anaconda IDE for machine learning: installing python modules/Packages like scikit-learn, Keras and Tensorflow etc.
  2. Programs involving pandas, Numpy and Scipy libraries.

1. Programs for classification

Build models using linear regression and logistic regression and apply it to classify a new instance

1. Write a program to demonstrate the following classifiers. Use an appropriate data set for building the model. Apply the model to classify a new instance.
2. Decision tree
3. K nearest neighbour
4. Naïve bayes
5. Support vector machine
6. Demonstration of Clustering algorithms using
   1. k-means
   2. Hierarchical algorithms (agglomerative etc).Interpret the clusters obtained.
7. Demonstrate ensemble techniques like boosting, bagging, random forests etc.
8. Build a classifier, compare its performance with an ensemble technique like random forest.
9. Evaluate various classification algorithms performance on a dataset using various measures like True Positive rate, False positive rate, precision, recall etc.
10. Demonstrate GA for optimization (minimization or maximization problem).
11. Case study on supervised/unsupervised learning algorithm
12. Installation of python environment/Anaconda IDE for machine learning: installing python modules/Packages like scikit-learn, Keras and Tensorflow etc.

#### Installation steps

Four Python 3.11 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary. The *offline installer* includes the components necessary for a default installation and only requires an internet connection for optional features.

After starting the installer, select “Install Now”. Python will be installed into your user directory. The Python Launcher for Windows will be installed according to the option at the bottom of the first page. The standard library, test suite, launcher and pip will be installed

If selected, the install directory will be added to your PATH.

#### Configuring Python

To run Python conveniently from a command prompt, set default environment variables in Windows.

To temporarily set environment variables, open Command Prompt and use the **set** command:

**C:\>set** PATH=C:\Program Files\Python 3.9;%PATH% **C:\>set** PYTHONPATH=%PYTHONPATH%;C:\My\_python\_lib **C:\>**python

#### Installing Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

#### pip install numpy

**Installing Pandas**

Pandas is an acronym that stands for " Python Data Analysis Library." Pandas is a widely used library for managing tabular data, data manipulation, and analysis based on NumPy.

It has functions for working with data sets such as analyzing, cleaning, exploring, and manipulating data. The fact that Pandas saves the information as a Python object containing rows and columns, comparable to data saved in Excel files, is probably its best feature

#### pip install pandas (or)

**pip3 install pandas**

#### Installing Scikit-learn (Sklearn)

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction for Python. This library is largely written in Python and built upon NumPy, SciPy and Matplotlib.

#### pip install -U scikit-learn

**Installing Keras**

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Keras allows you to switch between different back ends.

Out of these five frameworks, TensorFlow has adopted Keras as its official high-level API.

#### py -m venv keras

**Installing Tensorflow**

**TensorFlow** is a popular framework of **machine learning** and **deep learning**. It is **a free** and **open-source** library developed by **Google Brain Team**. It is used for numerical computation and data flow, which makes machine learning faster and easier.

TensorFlow can train and run the deep neural networks for image recognition, handwritten digit classification, recurrent neural network, **word embedding**, **natural language processing**, video detection, and many more. TensorFlow is run on multiple **CPU**s or **GPU**s and also mobile operating systems.

#### pip install tensorflow

1. Programs involving pandas, Numpy and Scipy libraries.

#load the library and check its version import numpy as np

np. version '1.12.1'

#create a list comprising numbers from 0 to 9 L = list(range(10))

#converting integers to string - this style of handling lists is known as list comprehension.

#List comprehension offers a versatile way to handle list manipulations tasks easily. We'll learn about them in future tutorials. Here's an example.

[str(c) for c in L]

['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

[type(item) for item in L]

[int, int, int, int, int, int, int, int, int, int]

#creating arrays np.zeros(10, dtype='int')

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

#creating a 3 row x 5 column matrix np.ones((3,5), dtype=float) array([[ 1., 1., 1., 1., 1.],

[ 1., 1., 1., 1., 1.],

[ 1., 1., 1., 1., 1.]])

#creating a matrix with a predefined value np.full((3,5),1.23)

array([[ 1.23, 1.23, 1.23, 1.23, 1.23],

[ 1.23, 1.23, 1.23, 1.23, 1.23],

[ 1.23, 1.23, 1.23, 1.23, 1.23]])

#create an array with a set sequence np.arange(0, 20, 2)

array([0, 2, 4, 6, 8,10,12,14,16,18])

#create an array of even space between the given range of values np.linspace(0, 1, 5)

array([ 0., 0.25, 0.5 , 0.75, 1.])

#create a 3x3 array with mean 0 and standard deviation 1 in a given dimension

np.random.normal(0, 1, (3,3))

array([[ 0.72432142, -0.90024075, 0.27363808],

[ 0.88426129, 1.45096856, -1.03547109],

[-0.42930994, -1.02284441, -1.59753603]])

#create an identity matrix np.eye(3)

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

#set a random seed np.random.seed(0)

x1 = np.random.randint(10, size=6) #one dimension

x2 = np.random.randint(10, size=(3,4)) #two dimension

x3 = np.random.randint(10, size=(3,4,5)) #three dimension

print("x3 ndim:", x3.ndim) print("x3 shape:", x3.shape) print("x3 size: ", x3.size) ('x3 ndim:', 3)

('x3 shape:', (3, 4, 5)) ('x3 size: ', 60)

#in a multidimensional array, we need to specify row and column index x2

array([[3, 7, 5, 5],

[0, 1, 5, 9],

[3, 0, 5, 0]])

#1st row and 2nd column value x2[2,3]

0

x = np.arange(10) x

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

#from start to 4th position x[:5]

array([0, 1, 2, 3, 4])

#from 4th position to end x[4:]

array([4, 5, 6, 7, 8, 9])

#return elements from first position step by two x[1::2]

array([1, 3, 5, 7, 9])

#reverse the array x[::-1]

array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

#You can concatenate two or more arrays at once. x = np.array([1, 2, 3])

y = np.array([3, 2, 1])

z = [21,21,21]

np.concatenate([x, y,z])

array([ 1, 2, 3, 3, 2, 1, 21, 21, 21])

#load library - pd is just an alias. I used pd because it's short and literally abbreviates pandas.

#You can use any name as an alias. import pandas as pd

#create a data frame - dictionary is used here where keys get converted to column names and values to row values.

data = pd.DataFrame({'Country': ['Russia','Colombia','Chile','Equador','Nigeria'],

'Rank':[121,40,100,130,11]})

data

|  |  |  |
| --- | --- | --- |
|  | **Country** | **Rank** |
| 0 | Russia | 121 |
| 1 | Colombia | 40 |
| 2 | Chile | 100 |
| 3 | Equador | 130 |
| 4 | Nigeria | 11 |

#We can do a quick analysis of any data set using: data.describe()

|  |  |
| --- | --- |
|  | **Rank** |
| count | 5.000000 |
| mean | 80.400000 |

|  |  |
| --- | --- |
|  | **Rank** |
| std | 52.300096 |
| min | 11.000000 |
| 25% | 40.000000 |
| 50% | 100.000000 |
| 75% | 121.000000 |
| max | 130.000000 |

#Among other things, it shows the data set has 5 rows and 2 columns with their respective names.

data.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 5 entries, 0 to 4

Data columns (total 2 columns):

Country 5 non-null object Rank 5 non-null int64 dtypes: int64(1), object(1) memory usage: 152.0+ bytes

#Let's create another data frame.

data = pd.DataFrame({'group':['a', 'a', 'a', 'b','b', 'b', 'c',

'c','c'],'ounces':[4, 3, 12, 6, 7.5, 8, 3, 5, 6]})

data

|  |  |  |
| --- | --- | --- |
|  | **group** | **ounces** |
| 0 | a | 4.0 |
| 1 | a | 3.0 |
| 2 | a | 12.0 |
| 3 | b | 6.0 |
| 4 | b | 7.5 |
| 5 | b | 8.0 |
| 6 | c | 3.0 |
| 7 | c | 5.0 |
| 8 | c | 6.0 |

#load the data

train = pd.read\_csv("~/Adult/train.csv") test = pd.read\_csv("~/Adult/test.csv") #check data set

train.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 32561 entries, 0 to 32560 Data columns (total 15 columns):

age 32561 non-null int64

workclass 30725 non-null object

fnlwgt 32561 non-null int64

education 32561 non-null object education.num 32561 non-null int64 marital.status 32561 non-null object occupation 30718 non-null object relationship 32561 non-null object race 32561 non-null object

sex 32561 non-null object capital.gain 32561 non-null int64 capital.loss 32561 non-null int64 hours.per.week 32561 non-null int64 native.country 31978 non-null object target 32561 non-null object dtypes: int64(6), object(9)

memory usage: 3.7+ MB

#load sklearn and encode all object type variables from sklearn import preprocessing

for x in train.columns:

if train[x].dtype == 'object':

lbl = preprocessing.LabelEncoder() lbl.fit(list(train[x].values))

train[x] = lbl.transform(list(train[x].values))

**2. Programs for classification**

# Build models using linear regression and logistic regression and apply it to classify a new instance

import numpy as np import pandas as pd

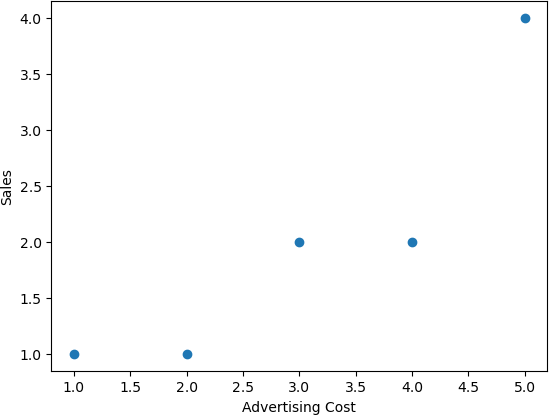
import matplotlib.pyplot as plt import statsmodels.api as sm

x = [1,2,3,4,5]

y = [1,1,2,2,4]

plt.scatter(x, y) plt.xlabel("Advertising Cost") plt.ylabel("Sales")

plt.show()



x1 = sm.add\_constant(x) results = sm.OLS(y,x1).fit()

results.summary()

OLS Regression Results

==================================================================

============

|  |  |
| --- | --- |
| Dep. Variable: y R-squared: | 0.817 |
| Model: OLS Adj. R-squared: | 0.756 |
| Method: Least Squares F-statistic: | 13.36 |
| Date: Wed, 21 Jun 2023 Prob (F-statistic): | 0.0354 |
| Time: 20:15:17 Log-Likelihood: | -3.3094 |
| No. Observations: 5 AIC: | 10.62 |
| Df Residuals: 3 BIC: | 9.838 |
| Df Model: 1 |  |
| Covariance Type: nonrobust |  |

==================================================================

============

coef std err t P>|t| [0.025 0.975]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| const | -0.1000 | 0.635 | -0.157 | 0.885 | -2.121 | 1.921 |
| x1 | 0.7000 | 0.191 | 3.656 | 0.035 | 0.091 | 1.309 |

slope =0.7

intercept = -0.1 def myfunc(x):

return slope \* x + intercept mymodel = list(map(myfunc, x))

print(slope) 0.7

print(intercept)

-0.1

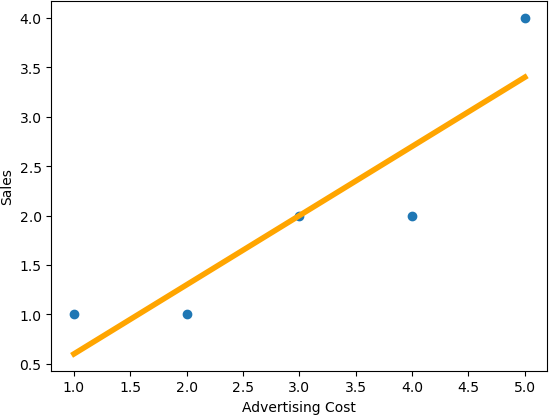
print(mymodel)

[0.6, 1.2999999999999998, 1.9999999999999996, 2.6999999999999997, 3.4]

plt.scatter(x,y)

fig = plt.plot(x,mymodel, lw=4, c='orange') plt.xlabel("Advertising Cost") plt.ylabel("Sales")

plt.show()



sales=myfunc(6) print(sales)

4.1

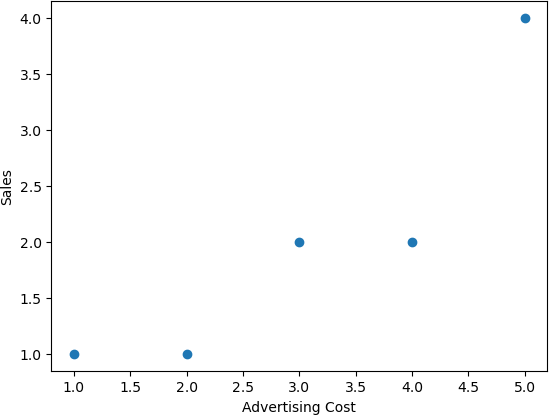
## Linear Regression Example 2

import matplotlib.pyplot as plt from scipy import stats

x = [1,2,3,4,5]

y = [1,1,2,2,4]

plt.scatter(x, y) plt.xlabel("Advertising Cost") plt.ylabel("Sales")

plt.show()

slope, intercept, r, p, std\_err = stats.linregress(x, y) def myfunc(x):

return slope \* x + intercept mymodel = list(map(myfunc, x))

print(slope) 0.7000000000000001

print(intercept)

-0.10000000000000009

print(mymodel)

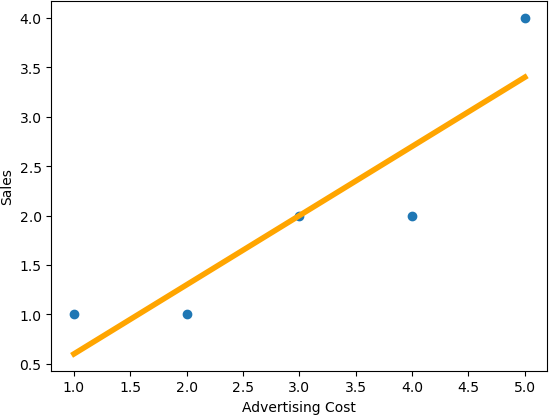
[0.6, 1.3, 2.0, 2.7, 3.4000000000000004]

print(std\_err) 0.19148542155126758

print(r) 0.903696114115064

plt.scatter(x, y) plt.xlabel("Advertising Cost") plt.ylabel("Sales")

plt.plot(x, mymodel, lw=4, c='orange') plt.show()



sales=myfunc(6) print(sales)

4.1

## Linear Regression example 3

import numpy as np import pandas as pd

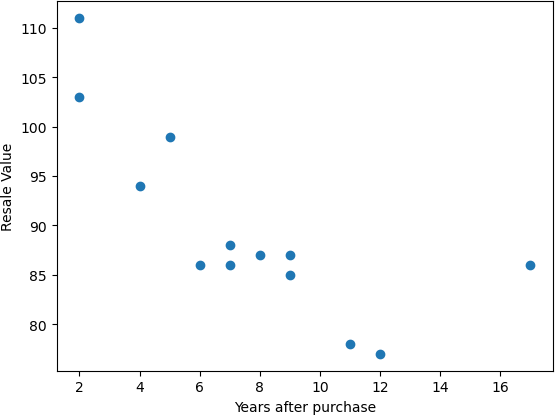
import matplotlib.pyplot as plt from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)

plt.xlabel("Years after purchase") plt.ylabel("Resale Value") plt.show()



slope, intercept, r, p, std\_err = stats.linregress(x, y) def myfunc(x):

return slope \* x + intercept mymodel = list(map(myfunc, x))

print(slope)

-1.7512877115526118

print(intercept) 103.10596026490066

print(mymodel)

[94.3495217071376, 90.84694628403238, 89.09565857247976, 90.84694628403238,

99.60338484179543, 73.33406916850626, 99.60338484179543, 87.34437086092716,

96.10080941869022, 83.84179543782193, 82.09050772626932, 87.34437086092716,

92.59823399558499]

print(std\_err) 0.453536157607742

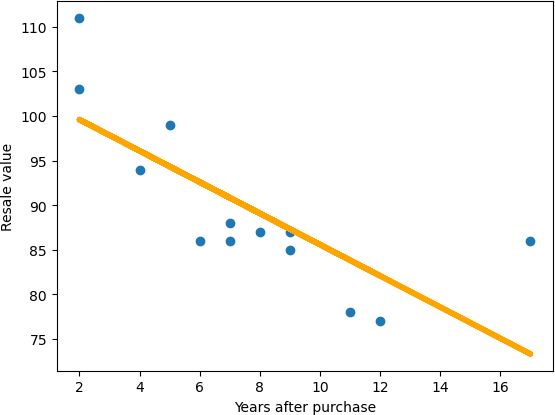
print(r)

-0.758591524376155

plt.scatter(x, y)

plt.xlabel("Years after purchase") plt.ylabel("Resale value")

plt.plot(x, mymodel, lw=4, c='orange') plt.show()



resale = myfunc(20) print(resale) 68.08020603384843

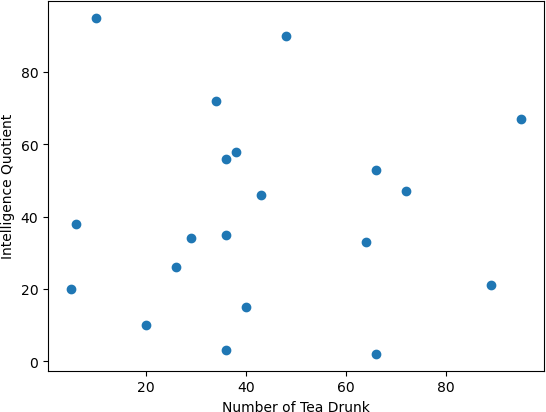
## Linear Regression example 4

import matplotlib.pyplot as plt from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]

y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

plt.scatter(x, y)

plt.xlabel("Number of Tea Drunk") plt.ylabel("Intelligence Quotient") plt.show()

slope, intercept, r, p, std\_err = stats.linregress(x, y) def myfunc(x):

return slope \* x + intercept mymodel = list(map(myfunc, x))

print(slope) 0.01391658139845263

print(intercept) 40.452282828936454

print(mymodel)

[94.3495217071376, 90.84694628403238, 89.09565857247976, 90.84694628403238,

99.60338484179543, 73.33406916850626, 99.60338484179543, 87.34437086092716,

96.10080941869022, 83.84179543782193, 82.09050772626932, 87.34437086092716,

92.59823399558499]

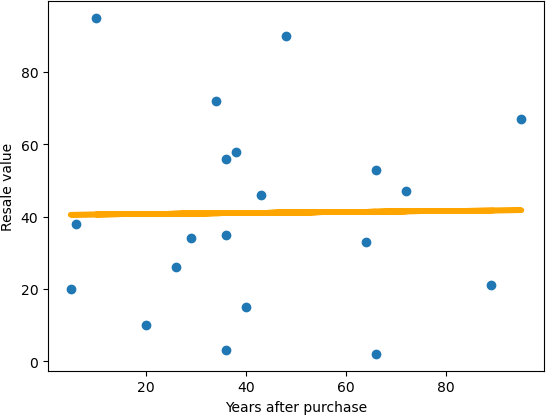
print(std\_err) 0.24627150586388075

print(r) 0.01331814154297491

plt.scatter(x, y)

plt.xlabel("Years after purchase") plt.ylabel("Resale value")

plt.plot(x, mymodel, lw=4, c='orange') plt.show()



#### Write a program to demonstrate the following classifiers. Use an appropriate data set for building the model. Apply the model to classify a new instance.

* 1. **Logistic regression**

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn import metrics

import seaborn as sns

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/logistic\_data.csv")

dataset

4.5192 2.6487 1.0

0 2.4443 1.5438 1.0

1 4.2409 1.8990 1.0

2 5.8097 2.4711 1.0

3 6.4423 3.3590 1.0

4 5.8097 3.2406 1.0

|  |  |  |
| --- | --- | --- |
| .. ... ... | ... |  |
| 94 5.9868 | 7.3641 | 0.0 |
| 95 4.6711 | 6.2592 | 0.0 |
| 96 7.5810 | 8.3703 | 0.0 |
| 97 4.6457 | 8.5676 | 0.0 |
| 98 4.6457 | 8.1676 | 0.0 |

[99 rows x 3 columns] # input

x = dataset.iloc[:, [0,1]].values x

Out:

array([[2.4443, 1.5438],

[4.2409, 1.899 ],

[5.8097, 2.4711],

[6.4423, 3.359 ],

[5.8097, 3.2406],

# target

y = dataset.iloc[:, 2].values y

Out:

array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

Out: LogisticRegression(random\_state=0)

classifier.classes\_ Out: array([0., 1.])

classifier.intercept\_

Out: array([3.12787746])

classifier.coef\_

Out: array([[ 1.51533124, -2.31207756]])

classifier.predict\_proba(xtest) Out[]:

array([[2.83459374e-05, 9.99971654e-01], [9.99431658e-01, 5.68342304e-04], [1.98901946e-03, 9.98010981e-01],

classifier.score(xtest, ytest)

Out[]: 1.0

y\_pred = classifier.predict(xtest) print(y\_pred)

[1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1.]

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

Accuracy : 1.0

cm = confusion\_matrix(ytest, y\_pred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[12 0]

[ 0 13]]

# Visualizing the Confusion Matrix

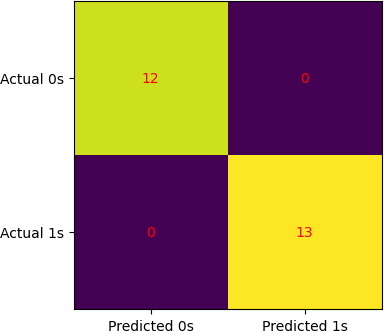
fig, ax = plt.subplots(figsize=(4, 4)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s')) ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s')) ax.set\_ylim(1.5, -0.5)

for i in range(2): for j in range(2):

ax.text(j, i, cm[i, j], ha='center', va='center', color='red') plt.show()



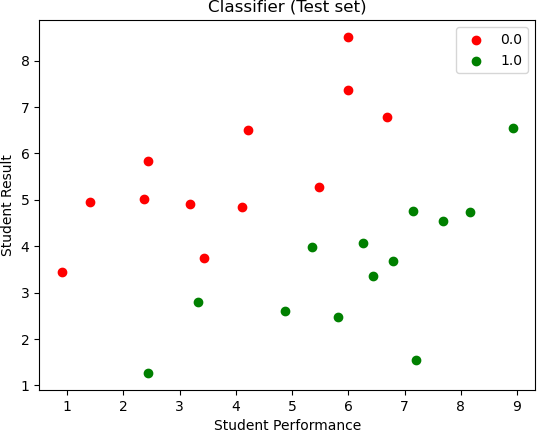
X\_set, y\_set = xtest, ytest

for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Student Performance') plt.ylabel('Student Result') plt.legend()

plt.show()



#Creates a decision boundary by predicting on a grid of values.

#Plots the boundary as a colored region (red for one class, green for the other).

#Overlays test set points on the boundary.  
X\_set[:, 0]

X\_set[:, 1]

X1, X2 = np.meshgrid(  
np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  
 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01)  
)

plt.contourf(  
X1, X2,   
classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
alpha = 0.75, cmap = ListedColormap(('red', 'green'))  
)   
plt.xlim(X1.min(), X1.max())

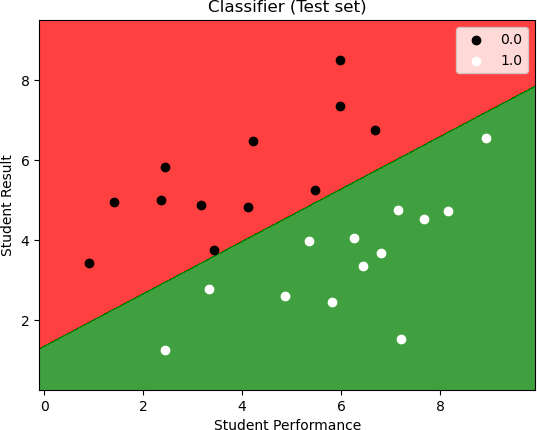
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('black', 'white'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Student Performance') plt.ylabel('Student Result') plt.legend()

plt.show()

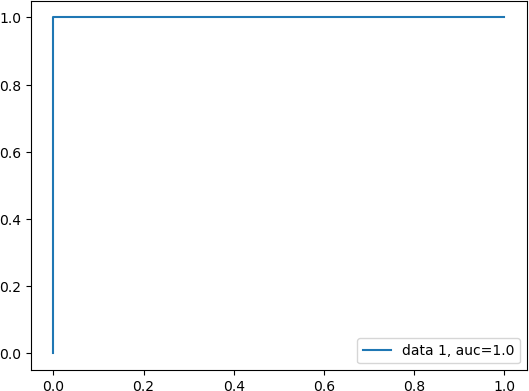


#Computes ROC curve, which shows how well the model distinguishes between classes.

#Plots the ROC curve with AUC (Area Under Curve) score.

y\_pred\_proba = classifier.predict\_proba(xtest)[::,1] fpr, tpr, \_ = metrics.roc\_curve(ytest, y\_pred\_proba) auc = metrics.roc\_auc\_score(ytest, y\_pred\_proba) plt.plot(fpr,tpr,label="data 1, auc="+str(auc)) plt.legend(loc=4)

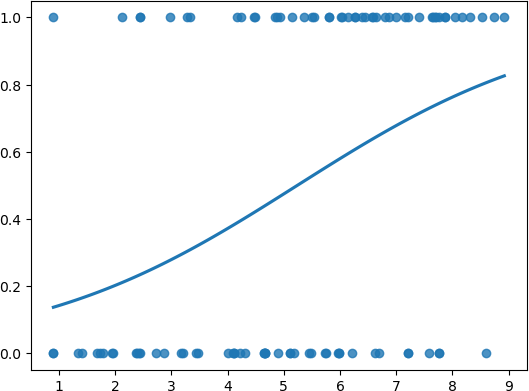
plt.show()



# Feature 1 vs Target

x1= dataset.iloc[:, [0]].values

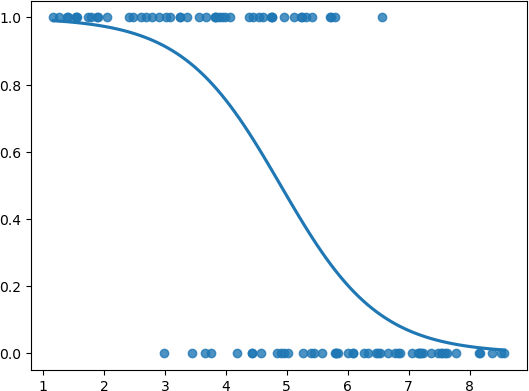
fig = sns.regplot(x=x1, y=y, data= dataset, logistic=True, ci=None)



# Feature 2 vs Target

x1= dataset.iloc[:, [1]].values

sns.regplot(x=x1, y=y, data= dataset, logistic=True, ci=None)



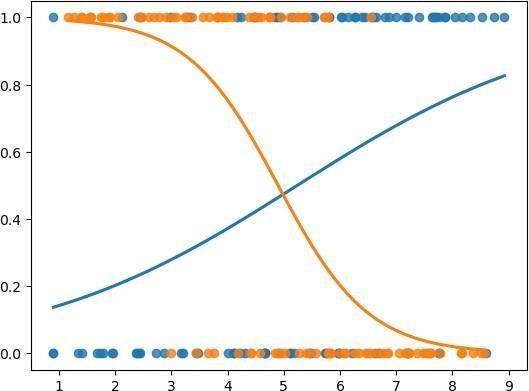
# Uses Seaborn's regplot() to visualize the logistic regression curve for each feature separately.

#Helps understand how each feature contributes to the classification.

x1= dataset.iloc[:, [0]].values

fig = sns.regplot(x=x1, y=y, data= dataset, logistic=True, ci=None) x1= dataset.iloc[:, [1]].values

sns.regplot(x=x1, y=y, data= dataset, logistic=True, ci=None)



## Multinomial Classification using Logistic Regression

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn import metrics

import seaborn as sns

dataset = pd.read\_csv("/Users/dianamoses/Downloads/logistic\_Iris.csv") dataset

Out[]:

5.1 3.5 1.4 0.2 Iris-setosa

|  |  |  |  |
| --- | --- | --- | --- |
| 0 4.9 3.0 | 1.4 | 0.2 | Iris-setosa |
| 1 4.7 3.2 | 1.3 | 0.2 | Iris-setosa |
| 2 4.6 3.1 | 1.5 | 0.2 | Iris-setosa |
| 3 5.0 3.6 | 1.4 | 0.2 | Iris-setosa |
| 4 5.4 3.9 | 1.7 | 0.4 | Iris-setosa |

.. ... ... ... ... ...

1. 6.7 3.0 5.2 2.3 Iris-virginica
2. 6.3 2.5 5.0 1.9 Iris-virginica
3. 6.5 3.0 5.2 2.0 Iris-virginica
4. 6.2 3.4 5.4 2.3 Iris-virginica
5. 5.9 3.0 5.1 1.8 Iris-virginica

# input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

# target

y = dataset.iloc[:, 4].values y

Out[68]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

…..'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

…..'Iris-virginica', 'Iris-virginica', 'Iris-virginica',], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain)

xtest = sc.transform(xtest)

classifier = LogisticRegression(random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: LogisticRegression(random\_state=0)

classifier.classes\_

Out[]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

classifier.intercept\_

Out[]: array([-0.3626655 , 1.68785188, -1.32518638])

classifier.coef\_ Out[]:

array([[-0.95553753, 1.15245455, -1.61438718, -1.55955728],

[ 0.47710429, -0.68052402, -0.19695366, -0.99093571],

[ 0.47843324, -0.47193053, 1.81134084, 2.55049299]])

classifier.score(xtest, ytest) Out[73]: 0.8421052631578947

y\_pred = classifier.predict(xtest) print(y\_pred)

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

['Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'

'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa'

'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'

'Iris-versicolor' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor']

Accuracy : 0.8421052631578947

cm = confusion\_matrix(ytest, y\_pred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[13 1 0]

[ 0 13 1]

[ 0 4 6]]

fig, ax = plt.subplots(figsize=(6, 6))

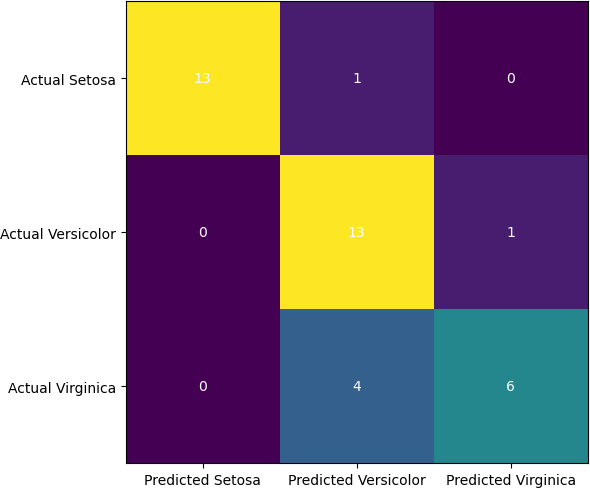
ax.imshow(cm) ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



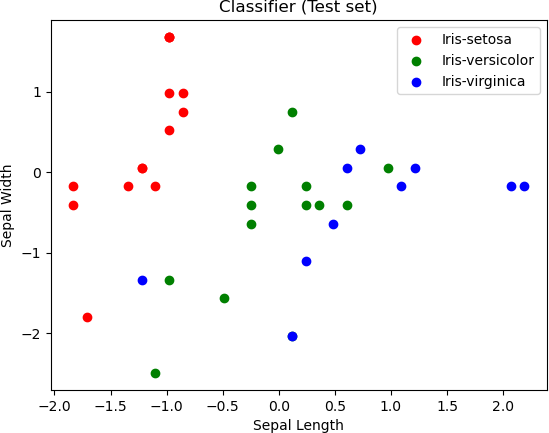
X\_set, y\_set = xtest, ytest

for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Sepal Length') plt.ylabel('Sepal Width') plt.legend()

plt.show()



## Decision tree-based Classification

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.metrics import accuracy\_score from sklearn.metrics import classification\_report from sklearn import metrics

import seaborn as sns

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Logistic\_Iris.csv")

dataset Out[]:

Sepal Length Sepal Width Petal Length Peatal Width Species

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns] # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

# target

y = dataset.iloc[:, 4].values y

Out[83]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

…. 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

…… 'Iris-virginica', 'Iris-virginica'], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

Tree induction using Gini Index

dtree\_gini = DecisionTreeClassifier(criterion = "gini", random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

dtree\_gini.fit(xtrain, ytrain)

Out[]: DecisionTreeClassifier(max\_depth=3, min\_samples\_leaf=5, random\_state=100)

y\_pred1 = dtree\_gini.predict(xtest) print("Predicted values:")

y\_pred1 Predicted values:

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',

'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica'], dtype=object)

accgini= accuracy\_score(ytest,y\_pred1)\*100 print ("\n\nAccuracy using Gini Index: ", accgini)

Accuracy using Gini Index: 89.47368421052632

cm = confusion\_matrix(ytest, y\_pred1)

print ("\n\n Confusion Matrix -using Gini Index: \n", cm)

Confusion Matrix -using Gini Index:

[[13 0 0]

[ 0 15 1]

[ 0 3 6]]

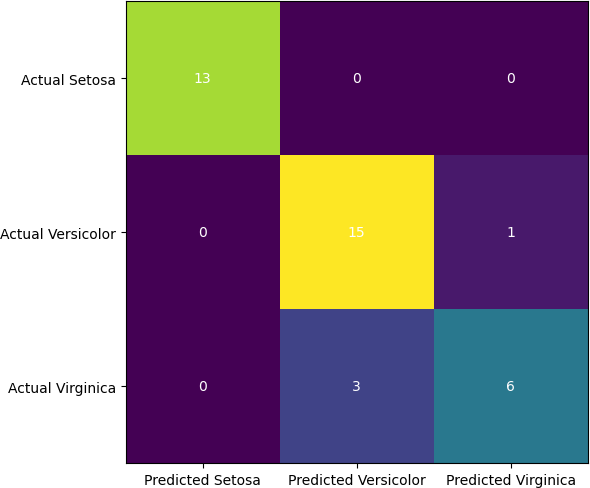
fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()

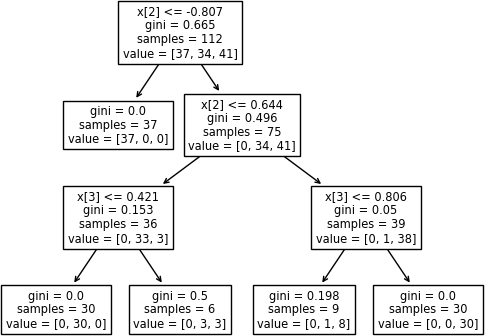
print("\n\nClassification Report – Using Gini Index: \n",classification\_report(ytest, y\_pred1))

Classification Report – Using Gini Index:

precision recall f1-score support

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iris-setosa | 1.00 |  | 1.00 | 1.00 | 13 |
| Iris-versicolor | 0.83 |  | 0.94 | 0.88 | 16 |
| Iris-virginica | 0.86 |  | 0.67 | 0.75 | 9 |
| Accuracy |  |  | 0.89 | 38 |  |
| macro avg | 0.90 | 0.87 | 0.88 | 38 |  |
| weighted avg | 0.90 | 0.89 | 0.89 | 38 |  |

tree.plot\_tree(dtree\_gini)



### Tree induction using Entropy

#using Entropy

dtree\_entropy = DecisionTreeClassifier(criterion = "entropy", random\_state = 100, max\_depth = 3, min\_samples\_leaf = 5)

dtree\_entropy.fit(xtrain, ytrain) y\_pred2 = dtree\_entropy.predict(xtest)

print("Predicted values:") print(y\_pred2)

Predicted values:

['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'

'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'

'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'

'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica']

acc\_entropy= accuracy\_score(ytest,y\_pred2)\*100 print ("\n\nAccuracy using Entropy: ", acc\_entropy)

Accuracy using Entropy: 89.47368421052632 cm = confusion\_matrix(ytest, y\_pred2)

print ("\n\n Confusion Matrix -using Entropy: \n", cm)

Confusion Matrix -using Entropy:

[[13 0 0]

[ 0 15 1]

[ 0 3 6]]

fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

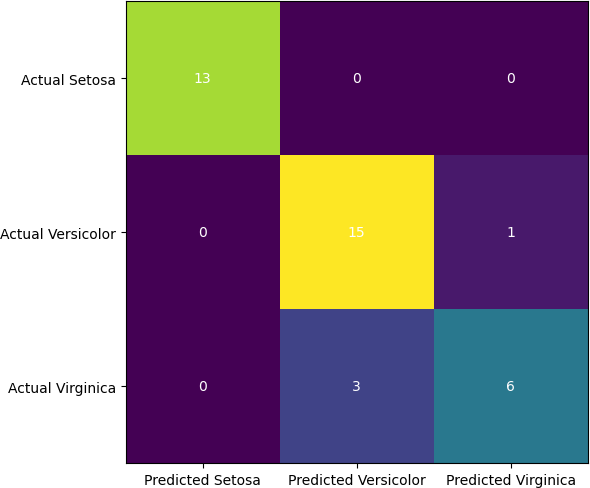
ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

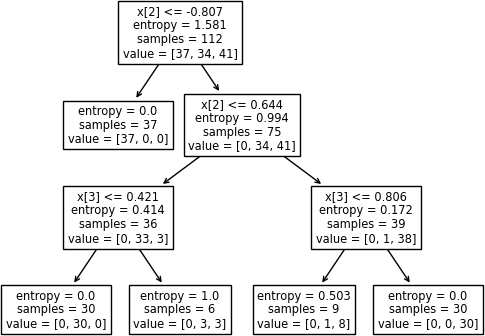
for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



print("Classification Report – Using Entropy: \n",classification\_report(ytest, y\_pred2)) Classification Report – Using Entropy:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iris-setosa | precision 1.00 | recall 1.00 | f1-score 1.00 | support 13 |
| Iris-versicolor 0.83 | | 0.94 | 0.88 | 16 |
| Iris-virginica 0.86 | | 0.67 | 0.75 | 9 |
| Accuracy |  |  | 0.89 | 38 |
| macro avg 0.90 | | 0.87 | 0.88 | 38 |
| weighted avg 0.90 | | 0.89 | 0.89 | 38 |
| tree.plot\_tree(dtree\_entropy) | | |  |  |



**Naïve bayes**

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Importing the dataset

dataset = pd.read\_csv('/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_car\_data.csv')

dataset Out[103]:

User ID Gender Age AnnualSalary Purchased

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 385 | Male | 35 | 20000 | 0 |
| 1 | 681 | Male | 40 | 43500 | 0 |
| 2 | 353 | Male | 49 | 74000 | 0 |
| 3 | 895 | Male | 40 | 107500 | 1 |
| 4 | 661 | Male | 25 | 79000 | 0 |

.. ... ... ... ... ...

995 863 Male 38 59000 0

996 800 Female 47 23500 0

997 407 Female 28 138500 1

998 299 Female 48 134000 1

999 687 Female 44 73500 0

[1000 rows x 5 columns] # input

x = dataset.iloc[:, [2, 3]].values x

Out[]:

|  |  |
| --- | --- |
| array([[ | 35, 20000], |
| [ | 40, 43500], |
| [ | 49, 74000], |
| ..., |  |
| [ | 28, 138500], |
| [ | 48, 134000], |
| [ | 44, 73500]]) |

# Target

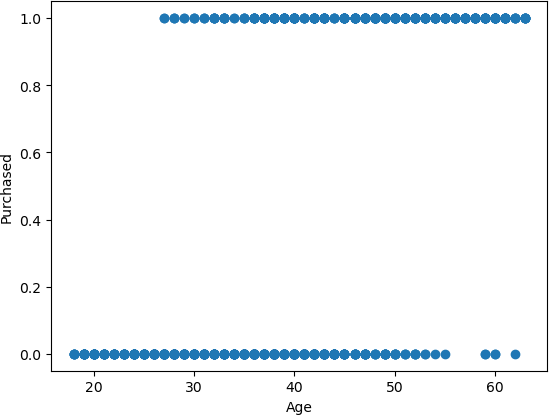
y = dataset.iloc[:, 4].values

y Out[]:

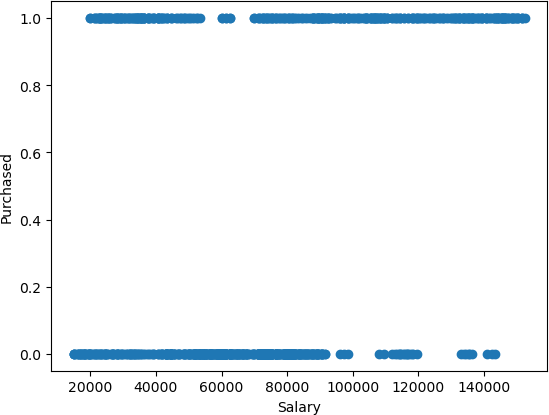
array([0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,

….. 0, 0, 0, 0, 1, 0, 0, 1, 1, 0])

x2 = dataset.iloc[:, [2]].values plt.scatter(x2,y) plt.xlabel("Age") plt.ylabel("Purchased") plt.show()



x3 = dataset.iloc[:, [3]].values plt.scatter(x3,y) plt.xlabel("Salary") plt.ylabel("Purchased") plt.show()



# Splitting the dataset into the Training set and Test set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

# Fitting Naive Bayes to the Training set classifier = GaussianNB() classifier.fit(x\_train, y\_train)

Out[]: GaussianNB()

# Predicting the Test set results y\_pred = classifier.predict(x\_test)

print("Predicted values:") print(y\_pred)

Predicted values:

[1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0]

acc= accuracy\_score(y\_test,y\_pred)\*100

print ("\n\nAccuracy of Naïve Bayes Classifier: ", acc)

Accuracy of Naïve Bayes Classifier: 88.0

# Making the Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred) cm

Out[]:

array([[140, 12],

[ 18, 80]])

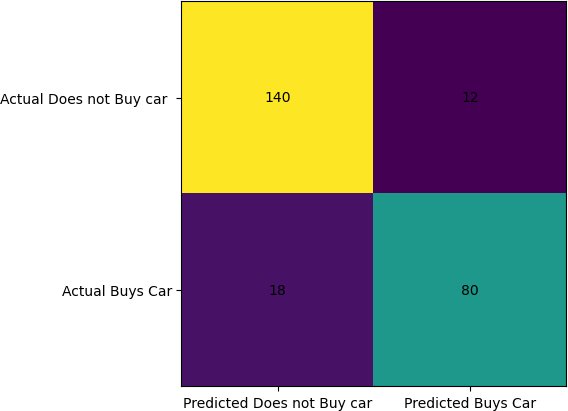
fig, ax = plt.subplots(figsize=(5, 5)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted Does not Buy car', 'Predicted Buys Car')) ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual Does not Buy car ', 'Actual Buys Car')) ax.set\_ylim(1.5, -0.5)

for i in range(2): for j in range(2):

ax.text(j, i, cm[i, j], ha='center', va='center', color='black') plt.show()



# Visualising the Training set results x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('white', 'black'))) mtp.xlim(X1.min(), X1.max())

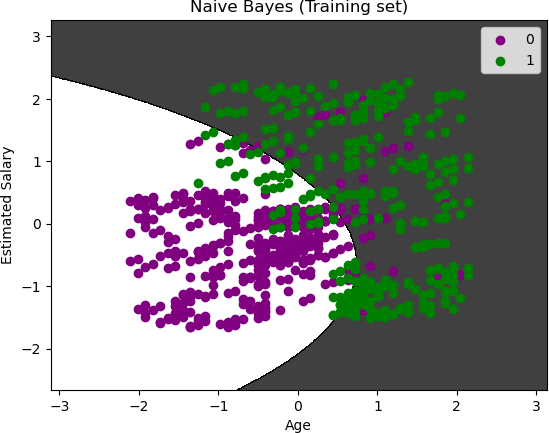
mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Naive Bayes (Training set)')

mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()



# Visualising the Test set results x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('white', 'black'))) mtp.xlim(X1.min(), X1.max())

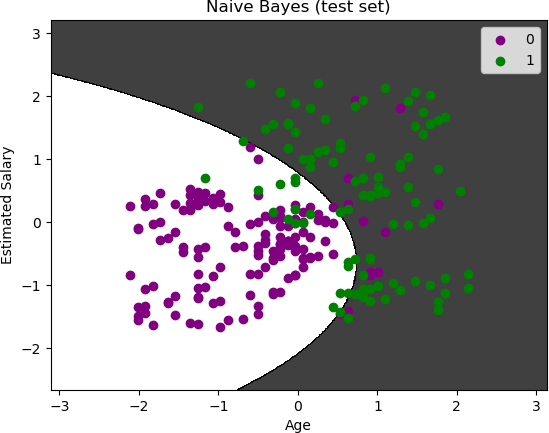
mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Naive Bayes (test set)')

mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()



Multinomial Classification using Naïve Bayes Classifier

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Logistic\_Iris.csv")

dataset.head

Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

# Splitting the dataset into the Training set and Test set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

# Fitting Naive Bayes to the Training set classifier = GaussianNB() classifier.fit(x\_train, y\_train)

Out[]: GaussianNB()

# Predicting the Test set results y\_pred = classifier.predict(x\_test)

print("Predicted values:") print(y\_pred)

Predicted values:

['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'

……..]

acc= accuracy\_score(y\_test,y\_pred)\*100

print ("\n\nAccuracy of Naïve Bayes Classifier: ", acc)

Accuracy of Naïve Bayes Classifier: 100.0 # Making the Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred) cm

Out[]:

array([[13, 0, 0],

[ 0, 16, 0],

[ 0, 0, 9]])

fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

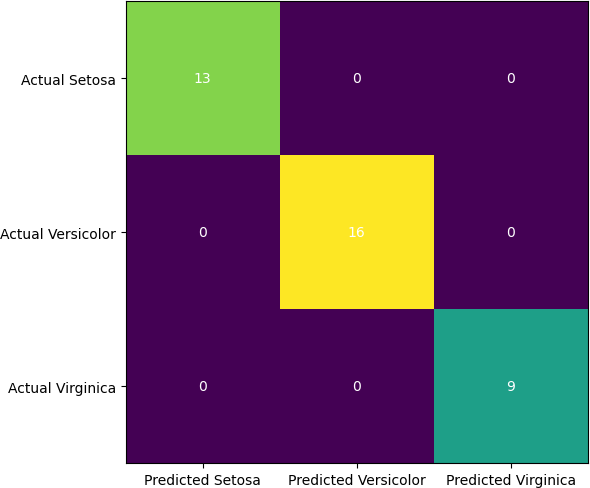
ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



K nearest neighbour

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.neighbors import KNeighborsClassifier

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_car\_data.csv")

dataset.head Out[]:

<bound method NDFrame.head of User ID Gender Age AnnualSalary Purchased

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 385 | Male | 35 | 20000 | 0 |
| 1 | 681 | Male | 40 | 43500 | 0 |
| 2 | 353 | Male | 49 | 74000 | 0 |
| 3 | 895 | Male | 40 | 107500 | 1 |
| 4 | 661 | Male | 25 | 79000 | 0 |

.. ... ... ... ... ...

995 863 Male 38 59000 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 996 | 800 | Female | 47 | 23500 | 0 |
| 997 | 407 | Female | 28 | 138500 | 1 |
| 998 | 299 | Female | 48 | 134000 | 1 |
| 999 | 687 | Female | 44 | 73500 | 0 |

[1000 rows x 5 columns]>

# input

x = dataset.iloc[:, [2,3]].values x

Out[]:

|  |  |
| --- | --- |
| array([[ | 35, 20000], |
| [ | 40, 43500], |
| [ | 49, 74000], |
| ..., |  |
| [ | 28, 138500], |
| [ | 48, 134000], |
| [ | 44, 73500]]) |

# output

y = dataset.iloc[:, 4].values y

Out[17]:

array([0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,

…….. 0, 0, 0, 0, 1, 0, 0, 1, 1, 0])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

knn = KNeighborsClassifier(n\_neighbors=7) knn.fit(xtrain, ytrain)

Out[]: KNeighborsClassifier(n\_neighbors=7)

ypred = knn.predict(xtest) print(ypred)

[1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 1]

knn.score(xtest, ytest)

Out[]: 0.924

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 92.4

cm = confusion\_matrix(ytest, ypred) print ("Confusion Matrix : \n", cm)

Confusion Matrix :

[[142 10]

[ 9 89]]

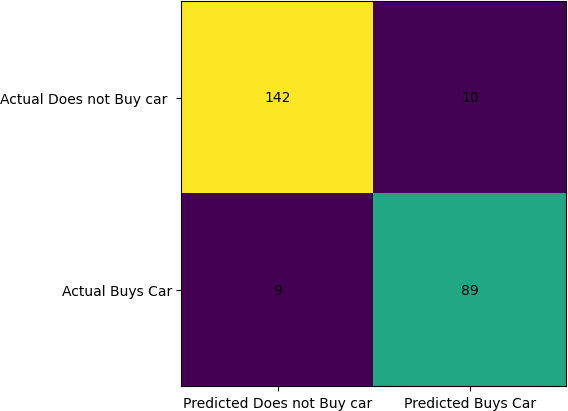
fig, ax = plt.subplots(figsize=(5, 5)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted Does not Buy car', 'Predicted Buys Car')) ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual Does not Buy car ', 'Actual Buys Car')) ax.set\_ylim(1.5, -0.5)

for i in range(2): for j in range(2):

ax.text(j, i, cm[i, j], ha='center', va='center', color='black') plt.show()



X\_set=x y\_set=y

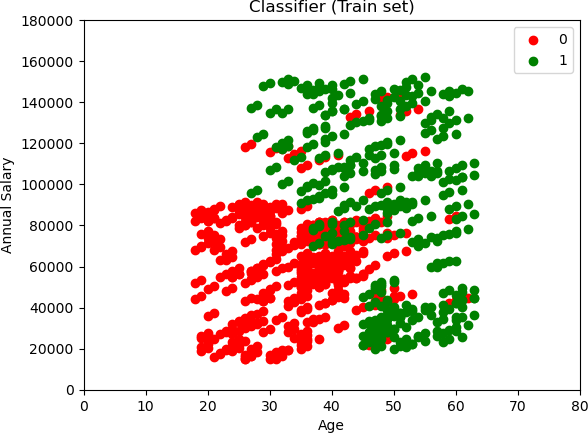
for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.xlim(0, 80)

plt.ylim(0, 180000) plt.title('Classifier (Train set)') plt.xlabel('Age') plt.ylabel('Annual Salary') plt.legend()

plt.show()



#Visulazing The Test Set X\_set=xtest

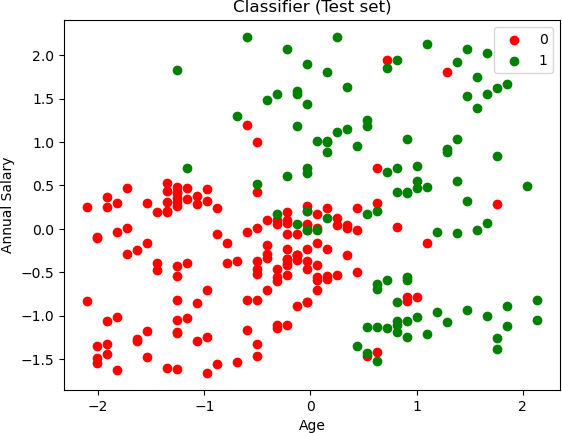
y\_set=ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Age') plt.ylabel('Annual Salary') plt.legend()

plt.show()



Multinomial Classification using KNN Classifier

import numpy as nm import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.neighbors import KNeighborsClassifier

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset

Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]>

# input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[31]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

……]])

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

…. 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

……. 'Iris-virginica'], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(xtrain, ytrain)

Out[]: KNeighborsClassifier(n\_neighbors=7)

ypred = knn.predict(xtest) print(ypred)

['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'

………….]

knn.score(xtest, ytest)

Out[]: 0.9736842105263158

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 97.36842105263158

cm = confusion\_matrix(ytest, ypred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

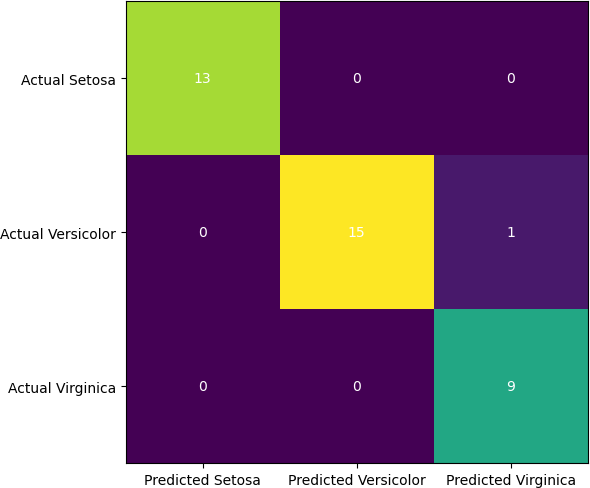
ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white')

plt.show()

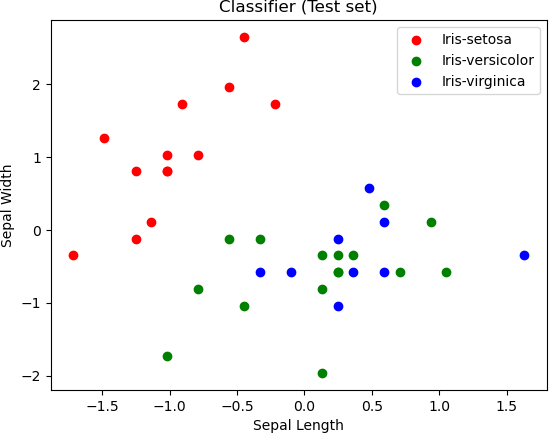
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Sepal Length') plt.ylabel('Sepal Width') plt.legend()

plt.show()



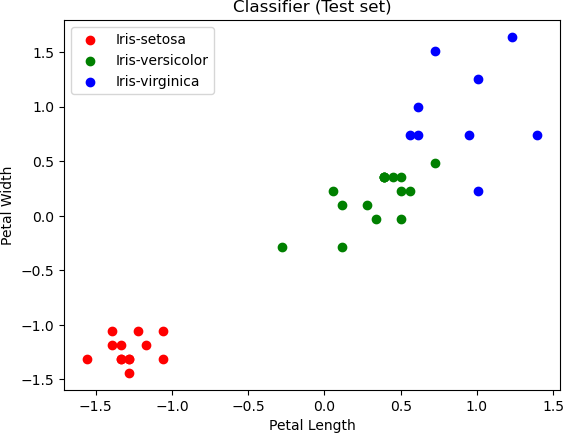
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 2], X\_set[y\_set == j, 3],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()

plt.show()



neighbors = nm.arange(1, 11) train\_accuracy = nm.empty(len(neighbors)) test\_accuracy = nm.empty(len(neighbors))

# Loop over K values

for i, k in enumerate(neighbors):

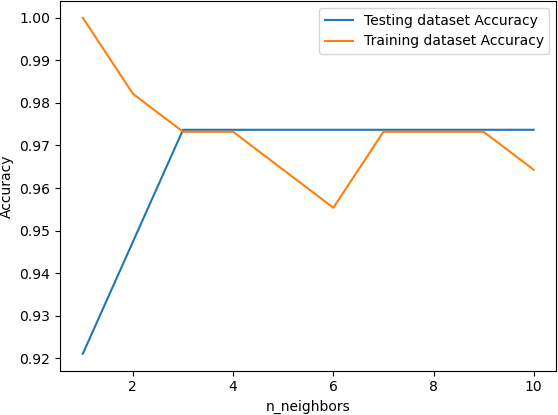
knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(xtrain, ytrain)

# Compute training and test data accuracy train\_accuracy[i] = knn.score(xtrain, ytrain) test\_accuracy[i] = knn.score(xtest, ytest)

# Generate plot

plt.plot(neighbors, test\_accuracy, label = 'Testing dataset Accuracy') plt.plot(neighbors, train\_accuracy, label = 'Training dataset Accuracy') plt.legend()

plt.xlabel('n\_neighbors') plt.ylabel('Accuracy') plt.show()



Support Vector Machine for Binomial Classification

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.svm import SVC

# Importing the dataset

dataset = pd.read\_csv('/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_car\_data.csv')

dataset.head

Out[]:

<bound method NDFrame.head of User ID Gender Age AnnualSalary Purchased

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 385 | Male | 35 | 20000 | 0 |
| 1 | 681 | Male | 40 | 43500 | 0 |
| 2 | 353 | Male | 49 | 74000 | 0 |
| 3 | 895 | Male | 40 | 107500 | 1 |
| 4 | 661 | Male | 25 | 79000 | 0 |

.. ... ... ... ... ...

995 863 Male 38 59000 0

996 800 Female 47 23500 0

997 407 Female 28 138500 1

998 299 Female 48 134000 1

999 687 Female 44 73500 0

[1000 rows x 5 columns]>

# input

x = dataset.iloc[:, [2,3]].values x

Out[]:

|  |  |
| --- | --- |
| array([[ | 35, 20000], |
| [ | 40, 43500], |
| [ | 49, 74000], |
| ..., |  |
| [ | 28, 138500], |
| [ | 48, 134000], |
| [ | 44, 73500]]) |

# output

y = dataset.iloc[:, 4].values y

Out[17]:

array([0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,

…….. 0, 0, 0, 0, 1, 0, 0, 1, 1, 0])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

classifier = SVC(kernel = 'linear', random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: SVC(kernel='linear', random\_state=0) ypred = classifier.predict(xtest)

print(ypred)

[1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0]

classifier.score(xtest, ytest)

Out[]: 0.84

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 84.0

cm = confusion\_matrix(ytest, ypred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[138 14]

[ 26 72]]

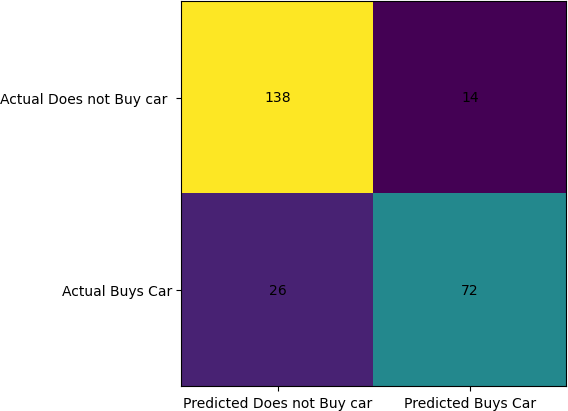
fig, ax = plt.subplots(figsize=(5, 5)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted Does not Buy car', 'Predicted Buys Car')) ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual Does not Buy car ', 'Actual Buys Car')) ax.set\_ylim(1.5, -0.5)

for i in range(2): for j in range(2):

ax.text(j, i, cm[i, j], ha='center', va='center', color='black') plt.show()



X\_set=x y\_set=y

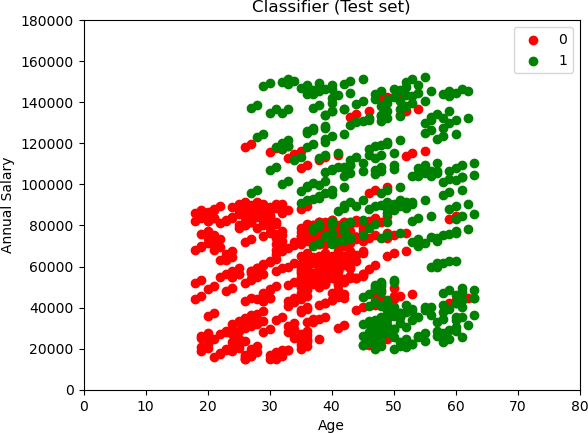
for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.xlim(0, 80)

plt.ylim(0, 180000) plt.title('Classifier (Test set)') plt.xlabel('Age') plt.ylabel('Annual Salary') plt.legend()

plt.show()



# Visualizing Test Results X\_set, y\_set = xtest, ytest

X1, X2 = nm.meshgrid(nm.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step =

0.01))

plt.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())

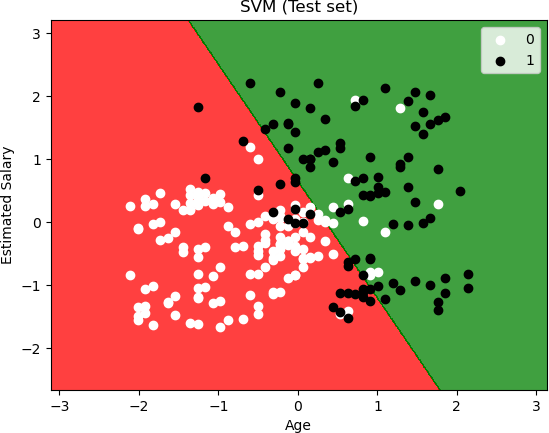
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('white', 'black'))(i), label = j) plt.title('SVM (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



## # USING RBF KERNAL FOR SVM

classifier = SVC(kernel = 'rbf', random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: SVC(random\_state=0)

ypred = classifier.predict(xtest) print(ypred)

[1 0 1 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 1 0]

classifier.score(xtest, ytest)

Out[]: 0.9

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 90.0

cm = confusion\_matrix(ytest, ypred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[138 14]

[ 11 87]]

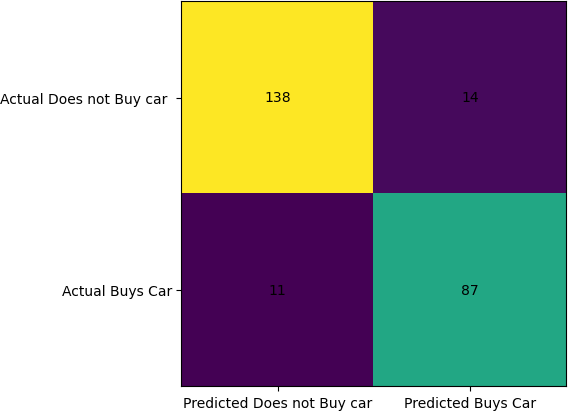
fig, ax = plt.subplots(figsize=(5, 5)) ax.imshow(cm)

ax.grid(False)

ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted Does not Buy car', 'Predicted Buys Car')) ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual Does not Buy car ', 'Actual Buys Car')) ax.set\_ylim(1.5, -0.5)

for i in range(2): for j in range(2):

ax.text(j, i, cm[i, j], ha='center', va='center', color='black') plt.show()



X\_set=x y\_set=y

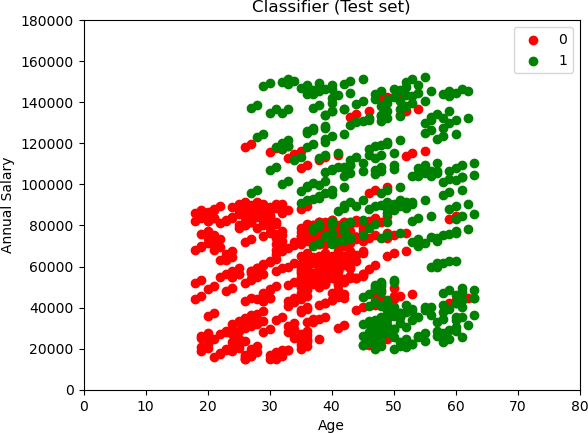
for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.xlim(0, 80)

plt.ylim(0, 180000) plt.title('Classifier (Test set)') plt.xlabel('Age') plt.ylabel('Annual Salary') plt.legend()

plt.show()



# Visualizing Test Results X\_set, y\_set = xtest, ytest

X1, X2 = nm.meshgrid(nm.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step =

0.01))

plt.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())

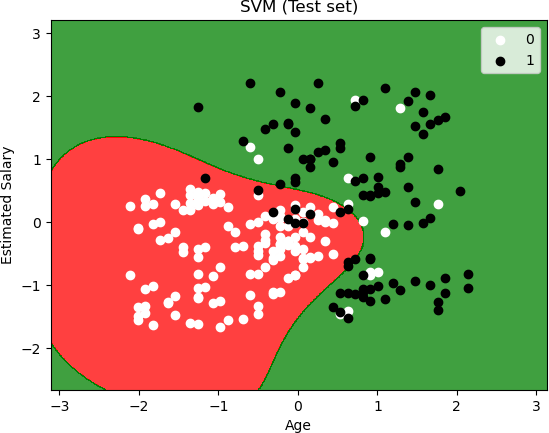
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('white', 'black'))(i), label = j) plt.title('SVM (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



Support Vector Machine for Multinomial Classification import numpy as nm

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.svm import SVC

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Logistic\_Iris.csv")

dataset.head

Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width Species

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]>

# input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

# Splitting the dataset into the Training set and Test set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

classifier = SVC(kernel = 'linear', random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: SVC(kernel='linear', random\_state=0) ypred = classifier.predict(xtest)

print(ypred)

['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'

……

'Iris-setosa' 'Iris-virginica']

classifier.score(xtest, ytest)

Out[]: 0.9736842105263158

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 97.36842105263158

cm = confusion\_matrix(ytest, ypred) print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

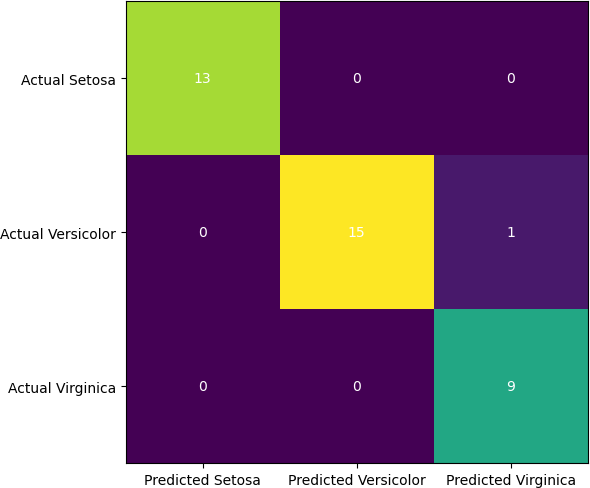
ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



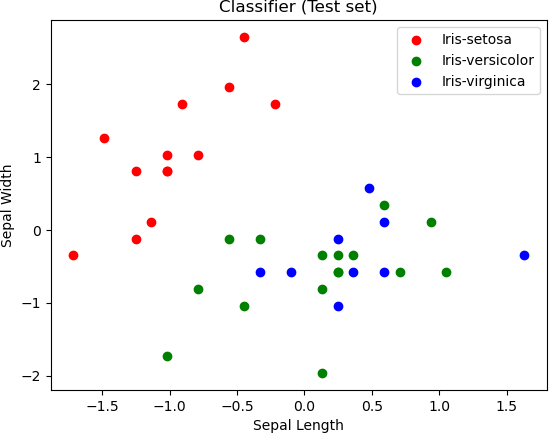
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Sepal Length') plt.ylabel('Sepal Width') plt.legend()

plt.show()



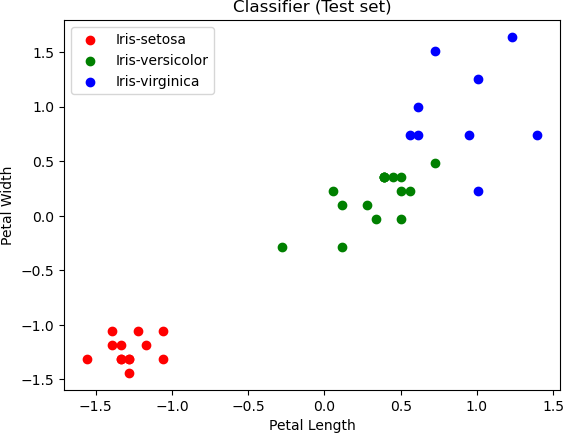
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 2], X\_set[y\_set == j, 3],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()

plt.show()



classifier = SVC(kernel = 'rbf', random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: SVC(random\_state=0)

ypred = classifier.predict(xtest) print(ypred)

['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'

…..

'Iris-setosa' 'Iris-virginica']

classifier.score(xtest, ytest)

Out[]: 0.9736842105263158

print ("\n\nAccuracy : ", accuracy\_score(ytest, ypred)\*100)

Accuracy : 97.36842105263158

cm = confusion\_matrix(ytest, ypred)

print ("Confusion Matrix : \n", cm) Confusion Matrix :

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

fig, ax = plt.subplots(figsize=(6, 6)) ax.imshow(cm)

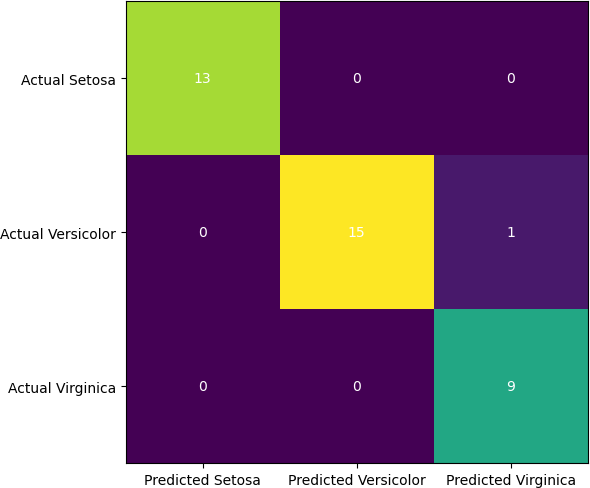
ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



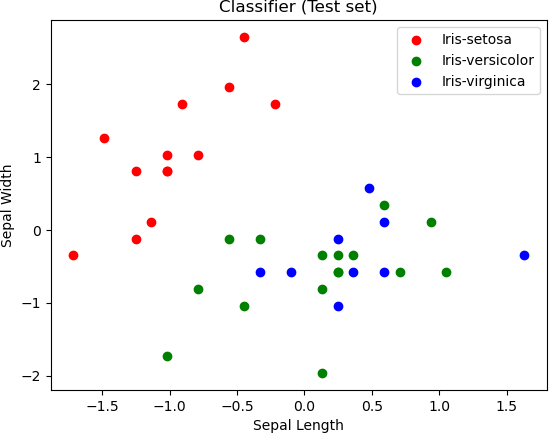
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Sepal Length') plt.ylabel('Sepal Width') plt.legend()

plt.show()



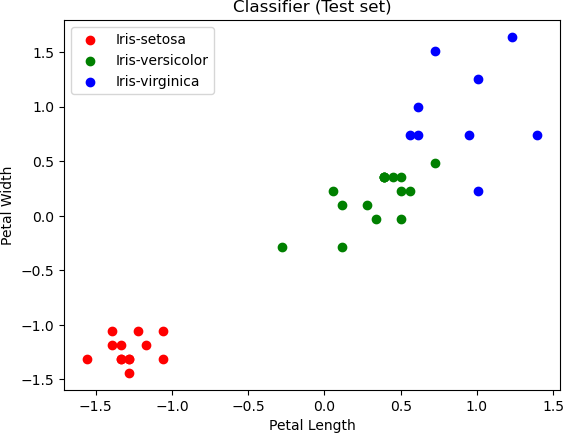
X\_set, y\_set = xtest, ytest

for i, j in enumerate(nm.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 2], X\_set[y\_set == j, 3],

c = ListedColormap(('red','green','blue'))(i), label = j)

plt.title('Classifier (Test set)') plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.legend()

plt.show()



Demonstration of Clustering algorithms using k-means import numpy as nm

import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.cluster import KMeans

from scipy.cluster.hierarchy import fcluster, linkage,dendrogram

import warnings warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Logistic\_Iris.csv")

dataset.head

Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

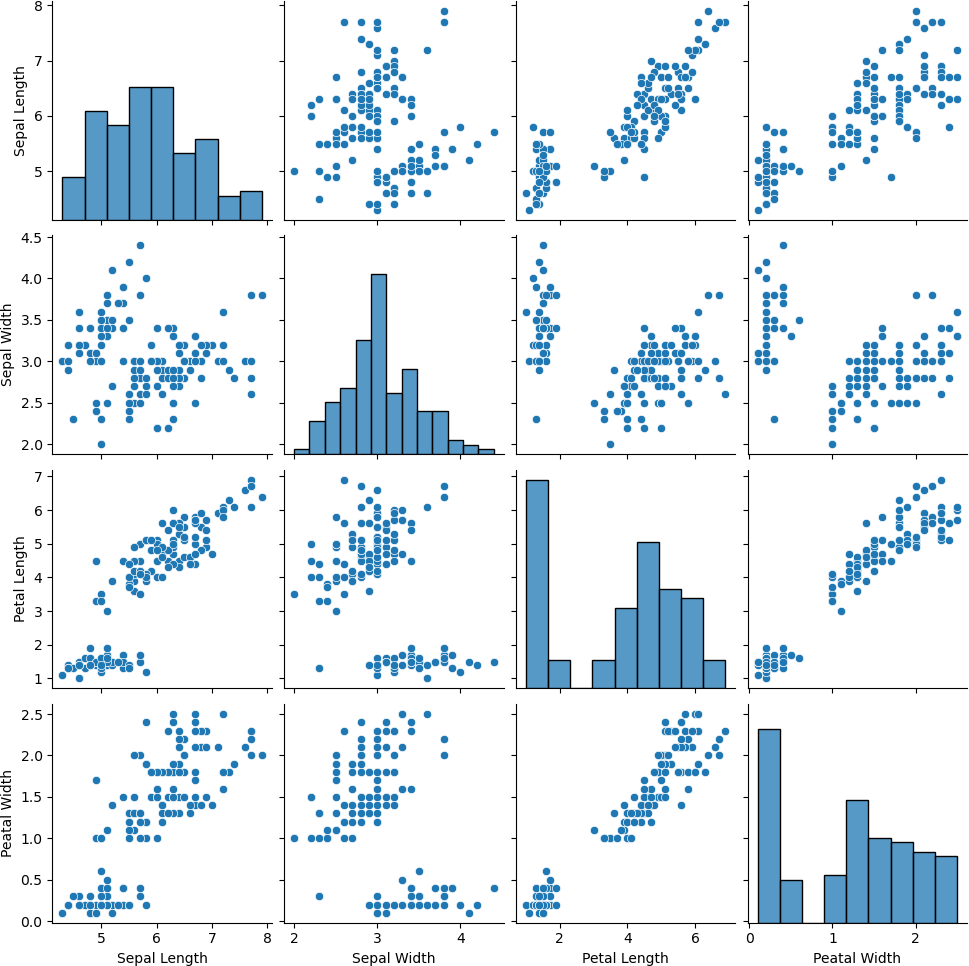
[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

sns.pairplot(dataset)



#Finding the optimum number of clusters for k-means classification Elbow = []

for i in range(1, 11):

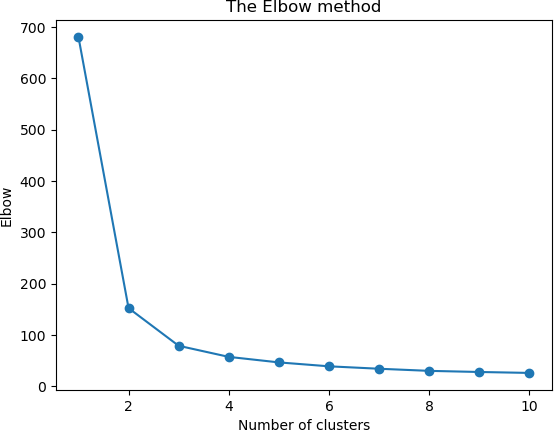
kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0) kmeans.fit(x)

Elbow.append(kmeans.inertia\_)

#Plotting the results onto a Line graph, allowing us to observe ‘The Elbow’ plt.plot(range(1, 11), Elbow, marker='o')

plt.title('The Elbow method') plt.xlabel('Number of clusters') plt.ylabel('Elbow')

#within cluster sum of squares plt.show()



kmeans=KMeans(n\_clusters=3, init='k-means++',max\_iter = 300, n\_init = 10, random\_state

= 0)

y\_kmeans=kmeans.fit\_predict(x)

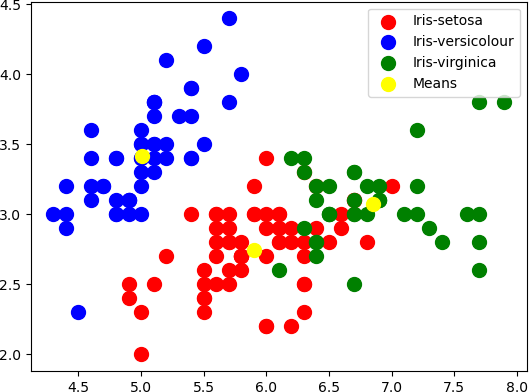
#Visualizing the Clusters

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa') plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s = 100, c= 'blue', label = 'Iris-versicolour')

plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c= 'green', label = 'Iris-virginica') #Plotting the centroids of the clusters

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'yellow', label = 'Means')

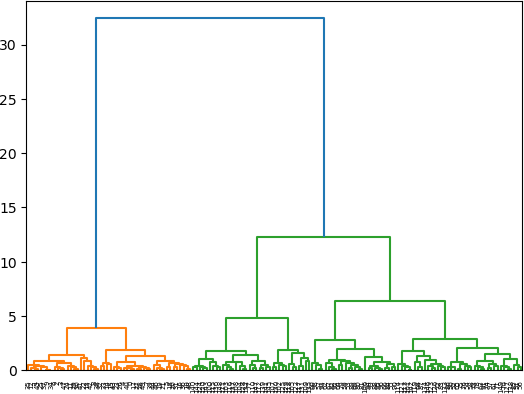
plt. legend()



distance\_matrix = linkage(x, method = 'ward', metric = 'euclidean') # Create a dendrogram

dn = dendrogram(distance\_matrix)

# Display the dendogram plt.show()



Demonstration of Clustering algorithms using Hierarchical algorithms (agglomerative etc).

import numpy as nm import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from matplotlib.colors import ListedColormap from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.cluster import KMeans

from scipy.cluster.hierarchy import fcluster, linkage,dendrogram

import warnings warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Logistic\_Iris.csv")

dataset.head

Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

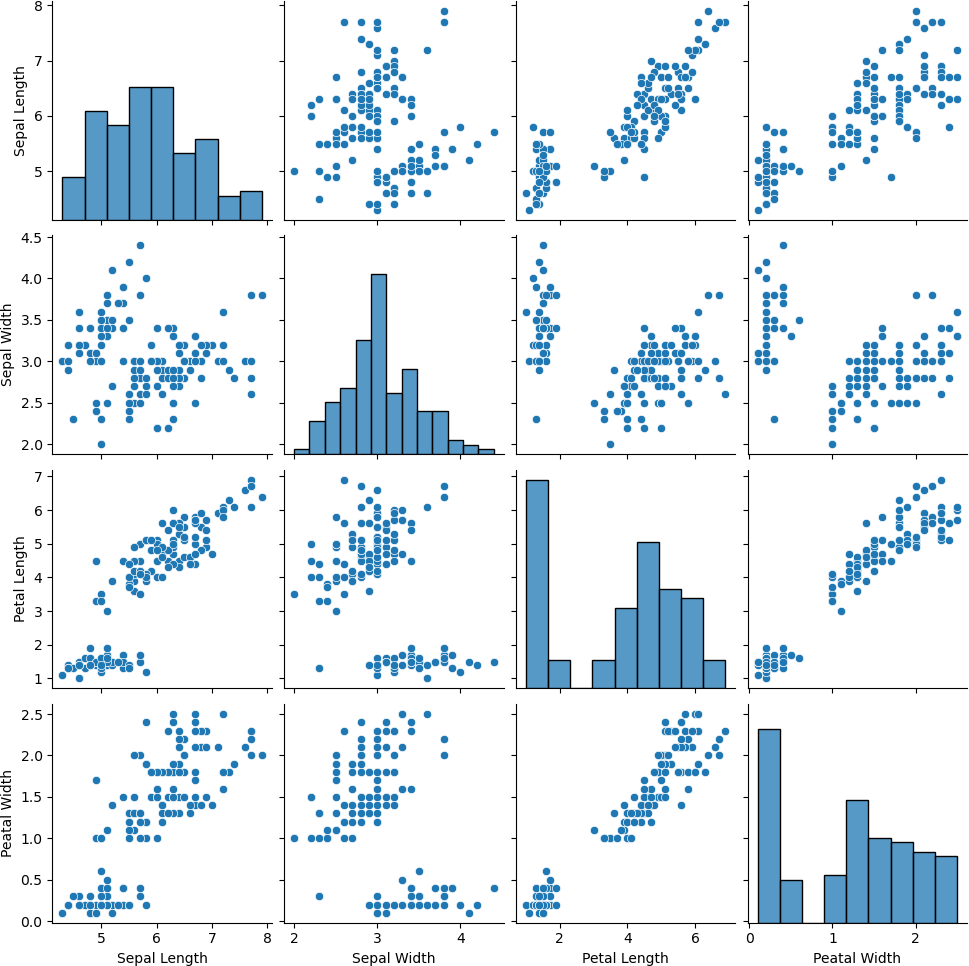
[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

sns.pairplot(dataset)

#Finding the optimum number of clusters for k-means classification Elbow = []

for i in range(1, 11):

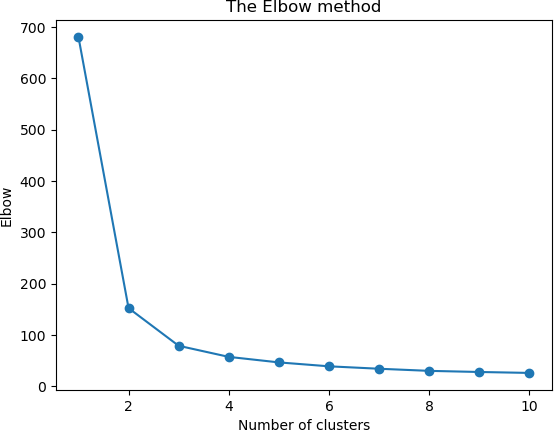
kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0) kmeans.fit(x)

Elbow.append(kmeans.inertia\_)

#Plotting the results onto a Line graph, allowing us to observe ‘The Elbow’ plt.plot(range(1, 11), Elbow, marker='o')

plt.title('The Elbow method') plt.xlabel('Number of clusters') plt.ylabel('Elbow')

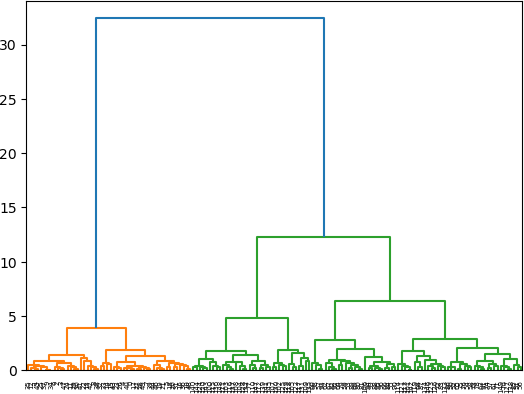
#within cluster sum of squares plt.show()



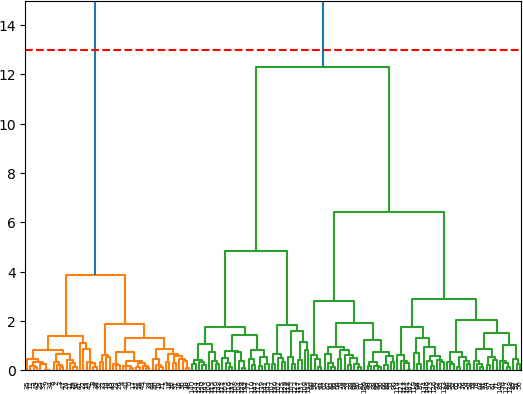
distance\_matrix = linkage(x, method = 'ward', metric = 'euclidean')

# Create a dendrogram

dn = dendrogram(distance\_matrix)

# Display the dendogram plt.show()

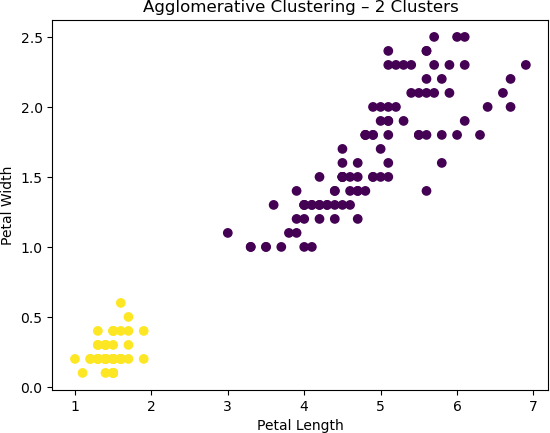
dn = dendrogram(distance\_matrix) plt.axhline(y=13, color='r', linestyle='--') plt.ylim(0,15)

plt.show()

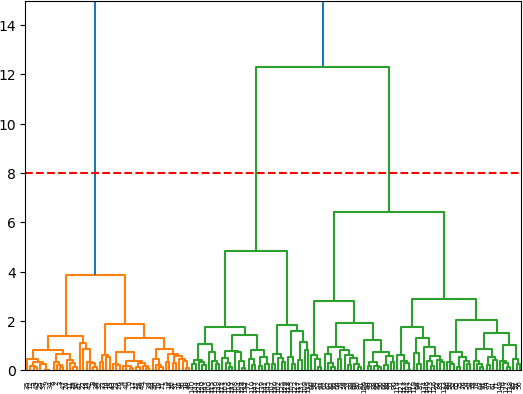
cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='ward') cluster.fit\_predict(x)

plt.title('Agglomerative Clustering – 2 Clusters') plt.scatter(x[:,2],x[:,3], c=cluster.labels\_, label= cluster.labels\_) plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

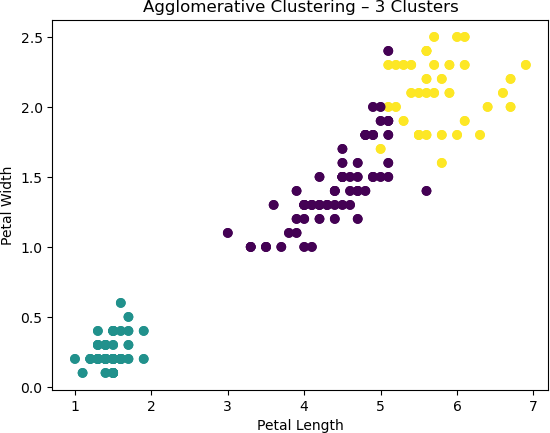


dn = dendrogram(distance\_matrix) plt.axhline(y=8, color='r', linestyle='--') plt.ylim(0,15)

plt.show()

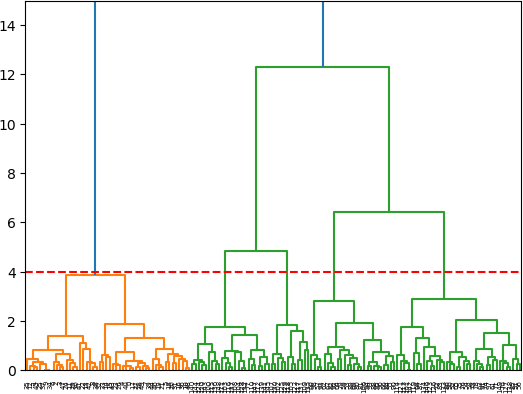
cluster = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='ward') cluster.fit\_predict(x)

plt.title('Agglomerative Clustering – 3 Clusters') plt.scatter(x[:,2],x[:,3], c=cluster.labels\_) plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

dn = dendrogram(distance\_matrix) plt.axhline(y=4, color='r', linestyle='--') plt.ylim(0,15)

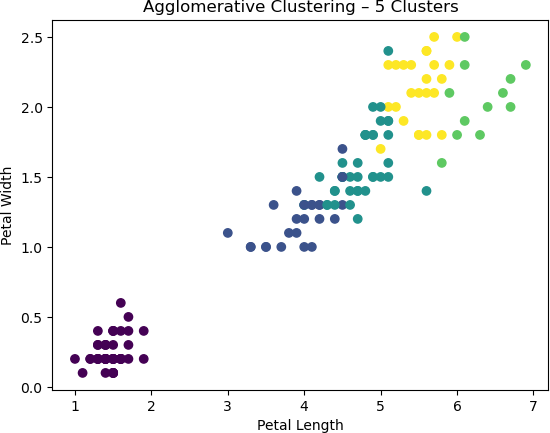
plt.show()



cluster = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward') cluster.fit\_predict(x)

plt.title('Agglomerative Clustering – 5 Clusters') plt.scatter(x[:,2],x[:,3], c=cluster.labels\_) plt.xlabel('Petal Length')

plt.ylabel('Petal Width')



**Demonstrate ensemble techniques like boosting, bagging, random forests etc.**

# Bagging

import numpy as nm import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn import model\_selection

from sklearn.ensemble import BaggingClassifier from sklearn.naive\_bayes import GaussianNB import warnings warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

# Splitting the dataset into the Training set and Test set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

Single = GaussianNB() Single.fit(xtrain, ytrain) Out[]: GaussianNB()

y\_pred = Single.predict(xtest)

print("Predicted values for single Naïve Bayes Classifier:") y\_pred

Predicted values for single Naïve Bayes Classifier: Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

….

'Iris-versicolor'], dtype='<U15')

Acc\_Single= accuracy\_score(ytest,y\_pred)\*100

print ("\n\nAccuracy using single Naïve Bayes Classifier: ",Acc\_Single)

Accuracy using single Naïve Bayes Classifier: 100.0 cm = confusion\_matrix(ytest, y\_pred)

print ("\n\n Confusion Matrix -using single Naïve Bayes Classifier: \n", cm)

Confusion Matrix -using single Naïve Bayes Classifier: [[13 0 0]

[ 0 16 0]

[ 0 0 9]]

# initialize the base classifier base\_cls = GaussianNB()

# no. of base classifier

num\_class = 100

# bagging classifier

Bag = BaggingClassifier(base\_estimator = base\_cls, n\_estimators = num\_class, random\_state

= 0)

Bag.fit(xtrain, ytrain)

Out[]: BaggingClassifier(base\_estimator=GaussianNB(), n\_estimators=100, random\_state=0)

results = model\_selection.cross\_val\_score(Bag, xtest, ytest, cv = 10) print("\n\nAccuracy using Bagged Set of Naïve Bayes Classifiers :", results.mean()\*100)

Accuracy using Bagged Set of Naïve Bayes Classifiers : 94.16666666666667

# Boosting

import numpy as nm import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn import model\_selection

from sklearn.ensemble import AdaBoostClassifier

import warnings warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain)

xtest = sc.transform(xtest)

adaboost = AdaBoostClassifier(n\_estimators = 50, learning\_rate = 0.2) adaboost. fit(xtrain, ytrain)

Out[]: AdaBoostClassifier(learning\_rate=0.2)

adaboost.score(xtest, ytest) Out[]: 0.8947368421052632

y\_pred = adaboost.predict(xtest)

print("Predicted values for AdaBoost Classifier:") y\_pred

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

…..

'Iris-virginica'], dtype=object)

Acc\_adaboost= accuracy\_score(ytest,y\_pred)\*100

print ("\n\nTest Accuracy using AdaBoost Classifier: ", Acc\_adaboost)

Test Accuracy using AdaBoost Classifier: 89.47368421052632 cm = confusion\_matrix(ytest, y\_pred)

print ("\n\n Confusion Matrix for AdaBoost Classifier: \n", cm)

Confusion Matrix for AdaBoost Classifier: [[13 0 0]

[ 0 15 1]

[ 0 3 6]]

fig, ax = plt.subplots(figsize=(6, 6))

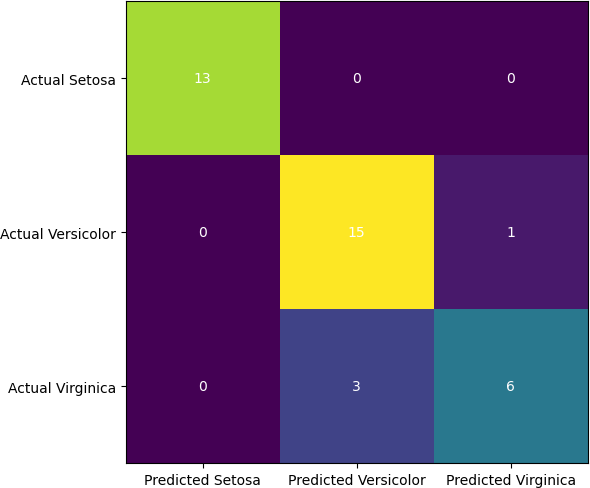
ax.imshow(cm) ax.grid(False)

ax.xaxis.set(ticks=(0,1,2), ticklabels=('Predicted Setosa', 'Predicted Versicolor', 'Predicted Virginica'))

ax.yaxis.set(ticks=(0,1,2), ticklabels=('Actual Setosa', 'Actual Versicolor', 'Actual Virginica')) ax.set\_ylim(2.5, -0.5)

for i in range(3): for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='white') plt.show()



# Random Forest Classifier

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier from sklearn import metrics

import seaborn as sns import warnings

warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

dtree= DecisionTreeClassifier() dtree.fit(xtrain, ytrain)

Out[]: DecisionTreeClassifier()

y\_pred1 = dtree.predict(xtest) print("Predicted values:") y\_pred1

Predicted values:

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica'], dtype=object)

acc\_dtree= accuracy\_score(ytest,y\_pred1)\*100

print ("\n\nAccuracy using Single Decision Tree: ", acc\_dtree)

Accuracy using Single Decision Tree: 97.36842105263158 cm = confusion\_matrix(ytest, y\_pred1)

print ("\n\n Confusion Matrix for Single Decision Tree: \n", cm)

Confusion Matrix for Single Decision Tree: [[13 0 0]

[ 0 15 1]

[ 0 0 9]]

# Create a Random forest Classifier

RF = RandomForestClassifier(n\_estimators = 100) # Train the model using the training sets RF.fit(xtrain, ytrain)

Out[]: RandomForestClassifier()

y\_pred2 = RF.predict(xtest) print("Predicted values:") y\_pred2

y\_pred2 = RF.predict(xtest) print("Predicted values:") y\_pred2

Predicted values:

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

…… 'Iris-virginica'], dtype=object)

acc\_rf= accuracy\_score(ytest,y\_pred2)\*100

print ("\n\nAccuracy using Random Forest: ", acc\_rf)

Accuracy using Random Forest: 97.36842105263158 cm = confusion\_matrix(ytest, y\_pred2)

print ("\n\n Confusion Matrix for Random Forest Classifier: \n", cm)

Confusion Matrix for Random Forest Classifier: [[13 0 0]

[ 0 15 1]

[ 0 0 9]]

# Evaluate various classification algorithms performance on a dataset using various measures like True Positive rate, False positive rate, precision, recall etc.

import numpy as nm import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap

from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import SVC

from sklearn.ensemble import BaggingClassifier from sklearn.metrics import accuracy\_score from sklearn.metrics import precision\_score from sklearn.metrics import recall\_score

import seaborn as sns

import warnings warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |

1. 5.9 3.0 5.1 1.8 Iris-virginica [150 rows x 5 columns]>

# input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0) sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

models = [] acc\_all = [] pres =[] tpr=[]

fpr=[]

# Logistic Regression

classifier = LogisticRegression(random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: LogisticRegression(random\_state=0)

y\_pred = classifier.predict(xtest)

acc= accuracy\_score(ytest,y\_pred)\*100 cm = confusion\_matrix(ytest, y\_pred) tp = sum(nm.diagonal(cm))

fp = cm[1,0] + cm[2,0]

fn = cm[0,1] + cm[0,2]

tn = cm.sum()-(tp + fp + fn)

# Precision = (TP / (TP/FP) pres1=tp / (tp + fp)

# Sensitivity, Recall, True Positive Rate = TP/P or TP / (TP+FN) tpr1 = tp / (tp + fn)

fpr1 = fp / (fp + tn)

models.append(classifier) acc\_all.append(acc) pres.append(pres1) tpr.append(tpr1) fpr.append(fpr1)

#For Naïve Bayes Classification classifier = GaussianNB() classifier.fit(xtrain, ytrain) Out[]: GaussianNB()

y\_pred = classifier.predict(xtest)

acc= accuracy\_score(ytest,y\_pred)\*100 cm = confusion\_matrix(ytest, y\_pred) tp = sum(nm.diagonal(cm))

fp = cm[1,0] + cm[2,0]

fn = cm[0,1] + cm[0,2]

tn = cm.sum()-(tp + fp + fn)

# Precision = (TP / (TP/FP) pres1=tp / (tp + fp)

# Sensitivity, Recall, True Positive Rate = TP/P or TP / (TP+FN) tpr1 = tp / (tp + fn)

fpr1 = fp / (fp + tn)

models.append(classifier) acc\_all.append(acc) pres.append(pres1) tpr.append(tpr1) fpr.append(fpr1)

#For Decision Tree based Classification classifier = DecisionTreeClassifier() classifier.fit(xtrain, ytrain)

Out[]: DecisionTreeClassifier()

y\_pred = classifier.predict(xtest)

acc= accuracy\_score(ytest,y\_pred)\*100 cm = confusion\_matrix(ytest, y\_pred) tp = sum(nm.diagonal(cm))

fp = cm[1,0] + cm[2,0]

fn = cm[0,1] + cm[0,2]

tn = cm.sum()-(tp + fp + fn)

# Precision = (TP / (TP/FP) pres1=tp / (tp + fp)

# Sensitivity, Recall, True Positive Rate = TP/P or TP / (TP+FN) tpr1 = tp / (tp + fn)

fpr1 = fp / (fp + tn)

models.append(classifier) acc\_all.append(acc) pres.append(pres1) tpr.append(tpr1) fpr.append(fpr1)

#For KNN Classification

classifier = KNeighborsClassifier(n\_neighbors=7) classifier.fit(xtrain, ytrain)

Out[]: KNeighborsClassifier(n\_neighbors=7)

y\_pred = classifier.predict(xtest)

acc= accuracy\_score(ytest,y\_pred)\*100 cm = confusion\_matrix(ytest, y\_pred) tp = sum(nm.diagonal(cm))

fp = cm[1,0] + cm[2,0]

fn = cm[0,1] + cm[0,2]

tn = cm.sum()-(tp + fp + fn)

# Precision = (TP / (TP/FP) pres1=tp / (tp + fp)

# Sensitivity, Recall, True Positive Rate = TP/P or TP / (TP+FN) tpr1 = tp / (tp + fn)

fpr1 = fp / (fp + tn)

models.append(classifier) acc\_all.append(acc) pres.append(pres1) tpr.append(tpr1) fpr.append(fpr1)

# For SVM Classification

classifier = SVC(kernel = 'rbf', random\_state = 0) classifier.fit(xtrain, ytrain)

Out[]: SVC(random\_state=0)

y\_pred = classifier.predict(xtest)

acc= accuracy\_score(ytest,y\_pred)\*100 cm = confusion\_matrix(ytest, y\_pred) tp = sum(nm.diagonal(cm))

fp = cm[1,0] + cm[2,0]

fn = cm[0,1] + cm[0,2]

tn = cm.sum()-(tp + fp + fn)

# Precision = (TP / (TP/FP) pres1=tp / (tp + fp)

# Sensitivity, Recall, True Positive Rate = TP/P or TP / (TP+FN) tpr1 = tp / (tp + fn)

fpr1 = fp / (fp + tn)

models.append(classifier) acc\_all.append(acc) pres.append(pres1) tpr.append(tpr1) fpr.append(fpr1)

print("\n Comparison of Classifiers \n ") print("\n Classifiers used\n")

models

Comparison of Classifiers

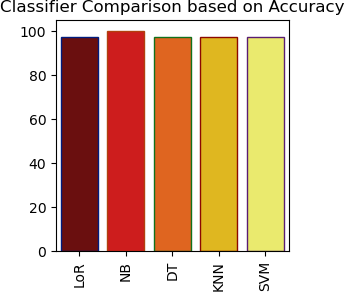
Classifiers used Out[]:

[LogisticRegression(random\_state=0), GaussianNB(), DecisionTreeClassifier(), KNeighborsClassifier(n\_neighbors=7), SVC(random\_state=0)]

plt.subplots(figsize=(3,3)) labels=['LoR', 'NB', 'DT', 'KNN', 'SVM']

sns.barplot(x=labels,y=acc\_all,palette='hot',edgecolor=sns.color\_palette('dark',7)) plt.xticks(rotation=90)

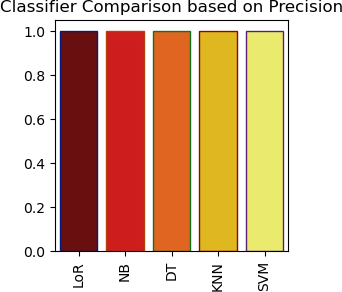
plt.title('Classifier Comparison based on Accuracy') plt.show()



plt.subplots(figsize=(3,3)) labels=['LoR', 'NB', 'DT', 'KNN', 'SVM']

sns.barplot(x=labels,y=pres,palette='hot',edgecolor=sns.color\_palette('dark',7)) plt.xticks(rotation=90)

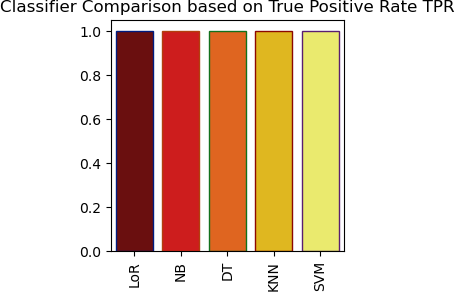
plt.title('Classifier Comparison based on Precision') plt.show()



plt.subplots(figsize=(3,3)) labels=['LoR', 'NB', 'DT', 'KNN', 'SVM']

sns.barplot(x=labels,y=tpr,palette='hot',edgecolor=sns.color\_palette('dark',7)) plt.xticks(rotation=90)

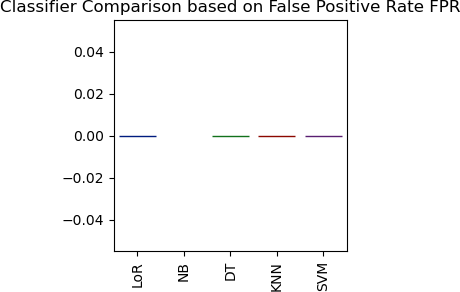
plt.title('Classifier Comparison based on True Positive Rate TPR') plt.show()



plt.subplots(figsize=(3,3)) labels=['LoR', 'NB', 'DT', 'KNN', 'SVM']

sns.barplot(x=labels,y=fpr,palette='hot',edgecolor=sns.color\_palette('dark',7)) plt.xticks(rotation=90)

plt.title('Classifier Comparison based on False Positive Rate FPR') plt.show()



from prettytable import PrettyTable table = PrettyTable()

table.title = 'Comparisson of Classifiers'

table.field\_names = ['Classifier', 'Accuracy', 'Precision', 'TPR', 'FPR'] table.add\_row(['LoR', acc\_all[0],pres[0],tpr[0],fpr[0]])

table.add\_row(['NB', acc\_all[1],pres[1],tpr[1],fpr[1]])

table.add\_row(['DT', acc\_all[2],pres[2],tpr[2],fpr[2]])

table.add\_row(['KNN', acc\_all[3],pres[3],tpr[3],fpr[3]])

table.add\_row(['SVM', acc\_all[4],pres[4],tpr[4],fpr[4]]) print(table)

+ +

| Comparisson of Classifiers |

+ + + + + +

| Classifier | Accuracy | Precision | TPR | FPR |

+ + + + + +

| LoR | 97.36842105263158 | 1.0 | 1.0 | 0.0 |

| NB | 100.0 | 1.0 | 1.0 | nan |

| DT | 97.36842105263158 | 1.0 | 1.0 | 0.0 |

| KNN | 97.36842105263158 | 1.0 | 1.0 | 0.0 |

| SVM | 97.36842105263158 | 1.0 | 1.0 | 0.0 |

+ + + + + +

**Build a classifier, compare its performance with an ensemble technique like random forest.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier from sklearn import metrics

import seaborn as sns import warnings

warnings.filterwarnings('ignore')

dataset = pd.read\_csv("/Users/dianamoses/Documents/MCET/Course Files/ML/ML LAB/Data/Logistic\_Iris.csv")

dataset.head Out[]:

<bound method NDFrame.head of Sepal Length Sepal Width Petal Length Peatal Width

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Species | | | | | |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| .. 145 | ...  6.7 | ...  3.0 | ... ..  5.2 | 2.3 | ...  Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

[150 rows x 5 columns]> # input

x = dataset.iloc[:, [0,1,2,3]].values x

Out[]:

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

………

# target

y = dataset.iloc[:, 4].values y

Out[]:

array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',

….

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

……

'Iris-virginica', 'Iris-virginica', 'Iris-virginica',

…..], dtype=object)

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

sc = StandardScaler()

xtrain = sc.fit\_transform(xtrain) xtest = sc.transform(xtest)

dtree= DecisionTreeClassifier() dtree.fit(xtrain, ytrain)

Out[]: DecisionTreeClassifier()

y\_pred1 = dtree.predict(xtest) print("Predicted values:") y\_pred1

Predicted values:

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica'], dtype=object)

acc\_dtree= accuracy\_score(ytest,y\_pred1)\*100

print ("\n\nAccuracy using Single Decision Tree: ", acc\_dtree)

Accuracy using Single Decision Tree: 97.36842105263158 cm = confusion\_matrix(ytest, y\_pred1)

print ("\n\n Confusion Matrix for Single Decision Tree: \n", cm)

Confusion Matrix for Single Decision Tree: [[13 0 0]

[ 0 15 1]

[ 0 0 9]]

# Create a Random forest Classifier

RF = RandomForestClassifier(n\_estimators = 100) # Train the model using the training sets RF.fit(xtrain, ytrain)

Out[]: RandomForestClassifier()

y\_pred2 = RF.predict(xtest) print("Predicted values:") y\_pred2

y\_pred2 = RF.predict(xtest) print("Predicted values:") y\_pred2

Predicted values:

Out[]:

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

…… 'Iris-virginica'], dtype=object)

acc\_rf= accuracy\_score(ytest,y\_pred2)\*100

print ("\n\nAccuracy using Random Forest: ", acc\_rf)

Accuracy using Random Forest: 97.36842105263158 cm = confusion\_matrix(ytest, y\_pred2)

print ("\n\n Confusion Matrix for Random Forest Classifier: \n", cm)

Confusion Matrix for Random Forest Classifier: [[13 0 0]

[ 0 15 1]

[ 0 0 9]]